Cibele Matos Freire

Complexidade Descritiva das Lógicas de Ordem Superior com Menor Ponto Fixo e Análise de Expressividade de Algumas Lógicas Modais

Fortaleza – CE

Agosto / 2010

Cibele Matos Freire

Complexidade Descritiva das Lógicas de Ordem Superior com Menor Ponto Fixo e Análise de Expressividade de Algumas Lógicas Modais

Dissertação de mestrado apresentada à coordenação do Programa de Mestrado e Doutorado em Ciência da Computação da Universidade Federal do Ceará como requisito para obtenção do título de mestre em Ciência da Computação.

Orientadora: Prof[a]. Dra. Ana Teresa Martins

LOGIA - Lógica e Inteligência Artificial
Mestrado e Doutorado em Ciência da Computação
Departamento de Computação
Universidade Federal do Ceará

Fortaleza – CE

Agosto / 2010

"A mente que se abre a uma nova idéia jamais voltará ao seu tamanho original."

Albert Einstein

Agradecimentos

Várias pessoas contribuíram para que esse trabalho fosse realizado. Sou profundamente grata a todas, em especial:

Aos meus pais, por todo amor que me deram e pelos ensinamentos e conselhos que sempre levarei comigo. Obrigada por sempre apoiarem as minhas escolhas e acreditarem em mim.

À minha orientadora Ana Teresa, por ter construído esse trabalho comigo e por sempre me incentivar e ajudar nos momentos em que precisei.

Aos meus amigos, que tornaram esses anos na UFC marcantes. Em particular, Camila e Karol, por serem braços abertos e ouvidos atentos às minhas angústias durante o período do mestrado. Ao Carlos, pelos conselhos e pelas discussões que me fazem refletir sobre a vida. Ao Francicleber e Alfredo, pelo companheirismo, pelas discussões e pelas conversas fiadas.

Ao meu noivo Tiago, pelo apoio durante os momentos de dificuldade, pela paciência, quando a minha já tinha acabado, e pelo amor, carinho e aconchego que me dá.

Às minhas irmãs, por todos os momentos de alegria, pelas implicâncias, pelas risadas e pelas confusões. :)

Resumo

Em Complexidade Descritiva investigamos o uso de lógicas para caracterizar classes problemas pelo viés da complexidade. Desde 1974, quando Fagin provou que NP é capturado pela lógica existencial de segunda-ordem, considerado o primeiro resultado da área, outras relações entre lógicas e classes de complexidade foram estabelecidas. Os resultados mais conhecidos normalmemte envolvem lógica de primeira-ordem e suas extensões, e classes de complexidade polinomiais em tempo ou espaço. Alguns exemplos são que a lógica de primeira-ordem estendida com o operador de menor ponto fixo captura a clsse P e que a lógica de segunda-ordem estendida com o operador de fecho transitivo captura a classe PSPACE. Nesta dissertação, analisaremos inicialmente a expressividade de algumas lógicas modais com relação ao problema de decisão REACH e veremos que é possível expressá-lo com as lógicas temporais CTL e CTL*. Analisaremos também o uso combinado de lógicas de ordem superior com o operador de menor ponto fixo e obteremos como resultado que cada nível dessa hierarquia captura cada nível da hierarquia determinística em tempo exponencial. Como corolário, provamos que a hierarquia de $\mathrm{HO}^i(\mathrm{LFP})$ não colapsa, ou seja, $\mathrm{HO}^i(\mathrm{LFP}) \subset \mathrm{HO}^{i+1}(\mathrm{LFP})$.

PALAVRAS-CHAVE: Complexidade Descritiva, Lógica Modal, Lógicas de Ordem Superior, Hierarquia Determinística de Tempo Exponencial, Operadores de Ponto Fixo.

Abstract

In Descriptive Complexity, we investigate the use of logics to characterize computational classes os problems through complexity. Since 1974, when Fagin proved that the class NP is captured by existential second-order logic, considered the first result in this area, other relations between logics and complexity classes have been established. Well-known results usually involve first-order logic and its extensions, and complexity classes in polynomial time or space. Some examples are that the first-order logic extended by the least fixed-point operator captures the class P and the second-order logic extended by the transitive closure operator captures the class PSPACE. In this dissertation, we will initially analyze the expressive power of some modal logics with respect to the decision problem REACH and see that is possible to express it with temporal logics CTL and CTL*. We will also analyze the combined use of higher-order logics extended by the least fixed-point operator and obtain as result that each level of this hierarchy captures each level of the deterministic exponential time hierarchy. As a corollary, we will prove that the hierarchy of $\mathrm{HO}^i(\mathrm{LFP})$, for $i \geq 2$, does not collapse, that is, $\mathrm{HO}^i(\mathrm{LFP}) \subset \mathrm{HO}^{i+1}(\mathrm{LFP})$.

KEYWORDS: Descriptive Complexity, Modal Logic, Higher-Order Logics, Deterministic Exponential Time Hierarchy, Fixed-Point Operators.

Sumário

1 Introdução p. 8

2 Complexidade p. 10

2.1 Complexidade Computacional . . . . . p. 10

2.2 Complexidade Descritiva . . . . . p. 16

3 Lógica Modal p. 22

3.1 Lógica Modal Básica . . . . . p. 22

3.2 As lógicas CTL* e CTL . . . . . p. 25

4 Lógicas com Operadores de Ponto Fixo p. 28

4.1 Operadores de Ponto Fixo . . . . . p. 28

4.2 Lógicas de Ponto Fixo . . . . . p. 31

5 Lógicas de Ordem Superior p. 36

5.1 Sintaxe e Semântica . . . . . p. 36

5.2 Expressividade . . . . . p. 39

6 Lógicas de Ordem Superior com Operador de Menor Ponto Fixo p. 43

6.1 Sintaxe e Semântica . . . . . p. 43

6.2 Expressividade . . . . . p. 44

7 Conclusão p. 48

Apêndice A – Lógicas K, S4 e S5 p. 50

Referências p. 53

1 Introdução

Ao investigarmos os fundamentos da Ciência da Computação nos deparamos com três questões centrais: O que é computar uma função, ou seja, como definir a noção de computação; Quais são os limites da computação, isto é, o que pode e o que não pode ser computado; Como medir a eficiência de um algoritmo para solucionar um problema. Enquanto as duas primeiras questões são respondidas no âmbito da Computabilidade [DSW94], a terceira é de interesse da Complexidade Computacional [Pap94].

Em Complexidade Computacional definimos a complexidade de um problema usando medidas como tempo e espaço. Por exemplo, um problema A é mais complexo que um problema B se o melhor algoritmo que o resolve requer mais tempo que o melhor algoritmo que resolve B. Da mesma forma que podemos dizer que um problema é mais complexo do que outro, podemos agrupá-los em conjuntos de problemas com a mesma complexidade, o que chamamos de classes de complexidade. Algumas classes bem conhecidas são P e NP, que são baseados em medidas de tempo, e PSPACE, que é baseada em medidas de espaço.

Apesar dessa abordagem para medir a complexidade de problemas ser bastante utilizada e possuir um apelo forte de engenharia, ela não é adequada do ponto do vista da matemática, pois não tratam de aspectos intrínsecos do problema e sim dos recursos físicos utilizados e dos métodos de resolução destes.

A área de Complexidade Descritiva, que está incluída na área de Teoria de Modelos Finitos, se propõe a caracterizar a complexidade de um problema por um viés independente da máquina, definindo classes de complexidade com base na linguagem lógica necessária para expressar os problemas e, além disso, relaciona tais classes com as classes de complexidade computacional. Desta forma, consideramos um problema mais complexo do que outro se ele necessita de uma linguagem mais poderosa para expressá-lo.

Um dos primeiros resultados da área foi provado por Fagin em 1974 e diz que os problemas da classe NP, problemas que podem ser resolvidos por uma máquina de Turing

não-determinística em tempo polinomial, são os mesmos que podem ser descritos pela lógica existencial de segunda-ordem ($\exists$SO). Esse resultado é importante, pois mostrou uma relação bastante estreita entre classes de complexidade computacional e descritiva, trazendo à tona a questão se é possível obter tais resultados quando consideramos outras classes, ou seja, será que para cada classe de complexidade computacional existe uma classe de complexidade descritiva que engloba os mesmos problemas.

Uma aplicação imediata da Complexidade Descritiva é na teoria de Banco de Dados. Um banco de dados relacional pode ser entendido como uma estrutura relacional finita e suas linguagens de consulta como extensões da lógica de primeira ordem. Sendo assim, muitas questões que envolvem a expressividade de linguagens de consulta em banco de dados podem ser analisadas pelas ferramentas da Complexidade Descritiva, pois na teoria de Banco de Dados não queremos apenas expressar a consulta, mas também saber se ela é computacionalmente viável.

Neste trabalho nos propomos a investigar a expressividade de algumas lógicas modais e das lógicas de ordem superior com o menor ponto fixo mostrando quais classes de complexidade computacional estariam relacionadas com as classes de complexidade descritivas definidas por elas. No capítulo 2 iniciamos com uma revisão de complexidade computacional e complexidade descritiva, apresentando definições e alguns resultados. Nossos primeiros resultados encontram-se no capítulo 3, onde investigamos o poder expressivo de algumas lógicas modais na especificação de consultas e mostramos que podemos expressar o problema REACH com as lógicas temporais CTL e CTL*. No capítulo 4 revisamos a adição de alguns operadores de ponto fixo à lógica de primeira ordem e vemos quão mais expressivas essas extensões se tornam. No capítulo 5 apresentamos as lógicas de ordem superior e discutimos que, apesar delas serem muito expressivas, não são suficientes para expressar todos os problemas computáveis. Finalmente, no capítulo 6 damos nossa contribuição ao analisar a expressividade das lógicas de ordem superior com menor ponto fixo e assim obter o resultado principal dessa dissertação, que mostra que a hierarquia determinística exponencial é capturada por essas lógicas. Os resultados desta disertação estão provados em [FM09] e [FM10].

2 Complexidade

Neste capítulo introduziremos algumas noções básicas e a notação que usaremos no decorrer do texto. Iniciaremos com uma breve revisão sobre complexidade computacional e depois introduziremos o tópico de complexidade descritiva apresentando algumas definições e teoremas importantes da área. As noções básicas de Computabilidade, Complexidade Computacional e Lógica usadas neste texto, inclusive os teoremas, lemas e corolários ,estão em [DSW94], [Pap94] e [EFT94], respectivamente.

2.1 Complexidade Computacional

Como vimos na Introdução, em complexidade computacional estamos interessados em analisar a dificuldade de problemas utilizando como parâmetro a quantidade de recursos necessários para resolvê-los. Os recursos podem ser tempo de computação, memória ou mesmo número de processadores, no caso de algoritmos distribuídos, dentre outros.

Uma vez que definimos um parâmetro para análise, podemos agrupar problemas com complexidade semelhante, de acordo com essa medida, para obter classes de complexidade. Para definir formalmente as classes de complexidade dois conceitos são fundamentais: problemas de decisão e máquinas de Turing.

Problemas de decisão são problemas formulados de modo que sua solução é simplesmente SIM ou NÃO. Por exemplo, o problema REACHABILITY, ou apenas REACH, é definido da seguinte forma: Dado um grafo G e dois vértices s e t, existe um caminho entre s e t em G? Uma observação interessante é que todo problema pode ser formulado como um problema de decisão [Pap94] e é por isso que eles são utilizados para definir as classes de complexidade.

Outra forma de ver um problema de decisão é como uma linguagem, ou um subconjunto de *strings*. A linguagem é definida da seguinte maneira. Primeiro fixamos um esquema de codificação para representar as instâncias do problema na forma de *strings*.

Podemos agora associar o problema de decisão ao subconjunto de *strings* que representam suas instâncias positivas. Por exemplo, no caso do problema REACH cada *string* representa um grafo e os dois vértices s e t. Portanto, REACH é associado ao subconjunto de *strings* que representam os grafos nos quais s alcança t. Assim, podemos tratar os problemas de decisão como linguagens, de modo que resolver um problema de decisão é o mesmo que reconhecer as *strings* de uma linguagem.

Máquinas de Turing são mecanismos que calculam funções sistematicamente: recebem uma entrada, realizam uma série de computações e eventualmente fornecem um resultado. Mais especificamente, a entrada é fornecida na forma de uma *string* escrita em uma fita, e a computação se desenrola com a máquina movimentando um cabeçote sobre essa fita, e ocasionalmente modificando o conteúdo de algumas posições. Cada movimento da máquina durante a computação é determinado pelo estado interno da máquina e pelo símbolo da fita que está sob o cabeçote nesse momento. Se eventualmente a máquina de Turing para, o resultado da computação pode ser o conteúdo escrito na fita, ou alguma indicação de SIM ou NÃO. Podemos formalizar essa noção de computação utilizando o conceito de função de transição. Essa noção de computação é capturada formalmente da seguinte maneira:

Definição 2.1.1. Uma máquina de Turing é uma quádrupla $M = (K, \Sigma, \delta, s)$, onde K é um conjunto finito de estados, $s \in K$ é o estado inicial, Σ é o alfabeto de M composto por um número finito de símbolos e δ é a função de transição. A função δ mapeia o conjunto $K \times \Sigma$ no conjunto $(K \cup \{h, yes, no\}) \times \Sigma \times (\{\rightarrow, \leftarrow, \_\})$. Uma transição $\delta(q_1, \sigma) = (q_2, \rho, D)$ significa que se M está no estado q_1 e o cabeçote está lendo o símbolo σ, então a máquina passará para o estado q_2, o símbolo ρ será escrito na fita e o cabeçote será movimentado na direção indicada por D. Os estados h (estado de parada), yes (estado de aceitação) e no (estado de rejeição) não pertencem ao conjunto K e a máquina para quando alcança um deles.

Definição 2.1.2. Dizemos que a máquina M aceita uma *string* de entrada x se ela para no estado de aceitação, o que denotaremos por $M(x) = yes$. A *string* x é rejeitada se a máquina para no estado de rejeição, $M(x) = no$. Dizemos que M decide uma linguagem $L \subseteq \Sigma^*$ se M aceita todas as *strings* de L e rejeita todas as *strings* que não estão em L.

Definição 2.1.3. Uma configuração de uma máquina M é uma tripla (q, x, y), onde $q \in K$ é um estado e x e y são *strings* em Σ^*. x é a *string* à esquerda do cabeçote, incluindo o símbolo que está sendo lido, y é a *string* à direita do cabeçote, possivelmente vazia, e q é o estado corrente.

Se M alcança o estado h então a computação termina e interpretamos o conteúdo da fita como o resultado da computação, o que denotaremos por $M(x) = y$. Usaremos a notação $M(x) \downarrow$ para indicar que a máquina parou e nos casos em que a máquina não para usaremos $M(x) \uparrow$.

A Definição 2.1.1 descreve a versão mais simples da máquina de Turing, mas existem diversas variações, como por exemplo máquinas com mais de uma fita, ou máquinas com restrições de leitura e escrita. Um resultado importante em teoria da computação é que essas variações não influeciam de forma significativa o poder de computação da máquina. Por exemplo, uma máquina com k fitas computa exatamente o mesmo conjunto de funções que a máquina com apenas uma. O ganho que pode haver é relacionado com a eficiência da computação, porém esse ganho é no máximo polinomial [Pap94]. Apesar disso, essas variações são bastante úteis, pois permitem construir máquinas mais simples para alguns problemas. Nesse trabalho usaremos máquinas de Turing com k fitas. Nesse caso a função de transição mapeia elementos de $K \times \Sigma^k$ no conjunto $(K \cup \{h, yes, no\}) \times (\Sigma \times \{\rightarrow, \leftarrow, \_\})^k$. Note que agora a transição é determinada pelos símbolos de todas as fitas, e que em cada passo da computação a máquina pode escrever e mover o cabeçote de todas as fitas.

Outra característica que podemos variar nas máquinas de Turing é o seu modo de computação. O modo de computação é o que define a forma como a máquina pode se mover e a sua noção de aceitação, ou seja, quando uma *string* deve ser aceita naquela máquina. Na máquina que definimos, para cada par composto por um símbolo e um estado, só há uma possibilidade de transição, ou seja, δ é uma função injetiva, e a computação dessa máquina pode ser descrita com uma linha com diversos pontos, onde cada ponto representa um estado e os pontos inicial e final seriam os estados inicial e final (*yes*,*no* ou h), respectivamente. Esse modo de computação é conhecido como modo *determinístico*.

Considere agora que δ não precisa ser uma função injetiva, mas que seja apenas uma relação. Nesse caso poderemos ter, para cada par de símbolo e estado, mais de uma possibilidade de movimentação e, quando isso acontece, dizemos que a computação se ramifica, ou seja, a máquina desenvolve computações em paralelo, uma para cada possibilidade de movimentação. Dessa forma, a computação pode ser representada por uma árvore, onde cada vértice será um estado, a raiz sendo o estado inicial, e os filhos de cada vértice serão os estados alcançáveis a partir do estado representado naquele vértice. Nesse modo, que chamamos *não-determinístico*, uma *string* é aceita quando pelo menos uma das folhas dessa árvore de computação é um estado de aceitação e rejeita caso contrário, ou seja, quando nenhuma das folhas é um estado de aceitação.

Além dos modos determinístico e não-determinístico, que são os mais conhecidos, apresentaremos o modo *alternante*. Nesse modo, o conjunto de estados é dividido em dois grupos, estados existenciais e estados universais, e a computação pode ser descrita por uma árvore, porém a noção de aceitação muda. De fato, essa noção de aceitação generaliza a noção de aceitação do modo não-determinístico, que é o caso particular do modo alternante no qual todos os estados são existenciais. A seguir definiremos formalmente a máquina de Turing alternante.

Definição 2.1.4. Uma *máquina de Turing alternante* é uma máquina de Turing M cujos estados são divididos em dois grupos: estados existenciais e estados universais. Seja x uma entrada para M e considere a árvore de computação de M. Defina recursivamente, começando das folhas e subindo em direção à configuração inicial, um subconjunto destas configurações, chamado de configurações de aceitação, como segue: Primeiro, toda configuração folha com estado 'sim' é de aceitação. Uma configuração C com estado universal é de aceitação se e somente se todas suas configurações sucessoras são de aceitação. Uma configuração C com estado existencial é de aceitação se e somente se pelo menos uma configuração sucessora é de aceitação. Finalmente, dizemos que M aceita x se a configuração inicial for de aceitação.

Agora que já sabemos o que são problemas de decisão e máquinas de Turing, podemos definir mais precisamente as classes de complexidade.

Como visto anteriormente, as classes de complexidade são conjuntos de problemas que tem complexidade semelhante. Para definir uma classe precisamos fixar os seguintes parâmetros: modelo de computação, modo de computação, recurso e função limitante. O modelo de computação que adotaremos é a máquina de Turing e o os modos de computação que usaremos podem ser o determinístico, o não-determinístico e o alternante. O recurso usualmente utilisado será tempo ou espaço e a função limitante será uma função $f : \mathbb{N} \to \mathbb{N}$ apropriada. Dizemos que uma função é apropriada quando ela é não-decrescente e existe uma máquina de Turing que, para uma entrada de tamanho n, computa a *string* $\sqcap^{f(n)}$ em tempo $O(n + f(n))$ e gastando $O(f(n))$ de espaço. Em outras palavras, só consideramos funções que podem ser computadas no tempo ou espaço que a própria função limita [Pap94].

Denotaremos por DTIME($f(n)$) o conjunto dos problemas que podem ser computados por uma máquina de Turing determinística em tempo $O(f(n))$. Analogamente, NTIME($f(n)$) e ASPACE($f(n)$) serão os conjuntos dos problemas computados por uma máquina de Turing não-determinística utilizando espaço $O(f(n))$ e por uma máquina de

Turing alternante utilizando espaço $O(f(n))$, respectivamente. Podemos também definir classes utilizando dois recursos simultaneamente. Por exemplo, ATIME-SPACE($f(n)$, $g(n)$) denota o conjunto de problemas computados em uma máquina alternante usando $O(f(n))$ tempo e $O(g(n))$ espaço. Em seguida veremos algumas classes de complexidade bastante conhecidas.

- L=DSPACE($\log n$) e NL=NSPACE($\log n$);
- P=$\bigcup_{k\in\mathbb{N}}$ DTIME(n^k) e NP=$\bigcup_{k\in\mathbb{N}}$ NTIME(n^k);
- PSPACE=$\bigcup_{k\in\mathbb{N}}$ DSPACE(n^k)=$\bigcup_{k\in\mathbb{N}}$ NSPACE(n^k)=NPSPACE;
- EXP=$\bigcup_{k\in\mathbb{N}}$ DTIME(2^{n^k}) e NEXP=$\bigcup_{k\in\mathbb{N}}$ NTIME(2^{n^k}).

Como toda linguagem tem seu complemento, é bastante natural que os problemas de decisão também tenham complementares. O complemento de um problema de decisão A, denotado por $\bar{\text{A}}$, é definido como o problema de decisão que responde SIM para as entradas que A responde NÃO e vice-versa. Dada uma classe de complexidade $\mathcal{C}$, definimos a classe complementar como sendo co$\mathcal{C}$=$\{\bar{\text{A}} \mid \text{A} \in \mathcal{C}\}$. Por exemplo, como SAT está em NP então UNSAT está em coNP.

Dentro de uma mesma classe de complexidade existem problemas com grau de dificuldade diferentes. Para dizer que um problema é pelo menos tão difícil do que outro utilizaremos a noção de redução. Dizer que um problema A pode ser reduzido a um problema B significa que é fácil computar A quando temos a resposta do problema B. Essa noção de redução será definida formalmente utilizando o conceito de máquina de Turing com oráculo. Essas máquinas são dotadas de um dispositivo especial que é capaz de responder instantaneamente SIM ou NÃO para um problema específico, por exemplo B, e utilizam essa informação para resolver outro problema, por exemplo A. Formalmente:

Definição 2.1.5. Uma máquina de Turing com oráculo $M^?$ é uma máquina de Turing com uma fita especial, chamada de fita de consulta, e três estados especiais, $q_?$ (estado de consulta), q_{yes} e q_{no}. Seja $M^?$ uma máquina e B uma problema de decisão, escrevemos M^B para denotar que M está equipada com um oráculo para B. Em algum momento da computação, M^B entrará no estado de consulta e, de acordo com o conteúdo da fita de consulta, passará ao estado q_{yes}, quando a *string* na fita de consulta pertencer a B, ou q_{no}, caso contrário.

Definição 2.1.6. Considere uma classe de complexidade $\mathcal{C}$ e dois problemas A e B. Dizemos que A é $\mathcal{C}$-redutível a B, A $\leq_{\mathcal{C}}$ B, se e somente se existe uma máquina de Turing com oráculo $M^?$ tal que M^B pertence a $\mathcal{C}$ e L(M^B) = A, ou seja, a linguagem decidida pela máquina é exatamente A.

Como dito anteriormente, queremos estabelecer uma relação entre os problemas de forma que seja possível afirmar quando um problema é pelo menos tão difícil quanto outro. Para a redução cumprir esse papel, $M^?$ deve pertencer a uma classe de complexidade fraca, como é o caso de P ou L. Dessa forma, sempre que nos referirmos a reduções estaremos falando de reduções em P ou L.

Dado que podemos relacionar os problemas de acordo com sua dificuldade, parece claro que podemos dizer quais problemas são os mais difíceis. Essa noção é dada mais precisamente abaixo.

Definição 2.1.7. Seja A um problema de decisão, $\mathcal{C}$ uma classe de complexidade e $\leq$ uma relação de redução. Dizemos que A é $\mathcal{C}$-difícil via a redução $\leq$ se, para todo B $\in \mathcal{C}$, B $\leq$ A. Se for $\mathcal{C}$-difícil e A $\in \mathcal{C}$ então dizemos que A é $\mathcal{C}$-completo.

Alguns exemplos de problemas completos bastante conhecidos são o SAT (NP-completo), o CVP (P-completo) e o REACH (NL-completo).

Além de comparar os problemas de uma classes, também podemos comparar as classes de complexidade entre si. Normalmente, queremos verificar, por exemplo, se aumentar o recurso utilizado ou mudar o modo de computação pode efetivamente aumentar o poder de computação da máquina, ou seja, ela será capaz de computar mais problemas. Um problema clássico de relação entre classes é a questão se P é igual a NP. Nesse caso estamos comparando o modo determinístico com o não-determinístico quando o recurso avaliado é tempo e, apesar de existir uma certa crença de que P$\neq$NP, o problema continua sem resposta.

Dessas relações entre as classes surge a noção de hierarquia. Nas hierarquias, as classes são organizadas em níveis e cada nível contém os níveis inferiores. Veremos a seguir a hierarquia polinomial, que é bastante conhecida, e a hierarquia logarítmica.

Definição 2.1.8. Considere a seguinte sequência de classes. Primeiro, Δ_0P = Σ_0P = Π_0P = P e, para todo $i \geq 0$,

1. $\Delta_{i+1}\mathrm{P} = \mathrm{P}^{\Sigma_i P}$

2. $\Sigma_{i+1}\mathrm{P} = \mathrm{NP}^{\Sigma_i P}$

3. $\Pi_{i+1}\mathrm{P} = \mathrm{coNP}^{\Sigma_i P}$.

Definimos a hierarquia polinomial como sendo a classe $\mathrm{PH} = \bigcup_{i \geq 0} \Sigma_i P$.

Definição 2.1.9. A *hierarquia de tempo logarítmica*, LH, é o conjunto dos problemas que são computados por uma máquina de Turing com alternância em ATIME-ALT[log $n, O(1)$], onde ATIME é como visto na seção 2.1 e ALT indica o número de alternâncias entre estados universais e existenciais em cada ramo da computação.

Em cada nível de PH temos problemas resolvidos por máquinas com oráculo que utilizam como oráculo problemas do nível inferior. Não sabemos se PH é uma hierarquia de fato, pois ainda não conseguimos garantir que a relação entre os níveis é estrita. Se for provado que P=NP, teremos que PH=P, ou seja, PH colapsa no primeiro nível. De fato, é difícil relacionar classes de diferentes tipos, como é o caso de P e NP, porém existem teoremas que estabelecem relações precisas quando as classes são do mesmo tipo. Mais precisamente, duas classes são do mesmo tipo se possuem o mesmo modo de computação e o mesmo recurso, variando apenas a função limitante. Um dos teoremas que relaciona classes do mesmo tipo é o seguinte:

Teorema 2.1.10. *[HS65]*(Teorema da Hierarquia de Tempo) *Se* $f(n) \geq n$ *é uma função apropriada, então a classe* DTIME($f(n)$) *está estritamente contida em* DTIME($f((2n+1)^3)$).

Esse resultado é muito importante, pois mostra que aumentar polinomialmente a quantidade de tempo no modo determinístico efetivamente aumenta o poder de computação das máquinas. Disso podemos provar que:

Corolário 2.1.11. P $\subset$ EXP*, ou seja,* P *é um subconjunto próprio de* EXP*.*

2.2 Complexidade Descritiva

Na Complexidade Descritiva o principal objetivo é caracterizar a complexidade de um problema através da lógica. Intuitivamente, queremos saber em quais classes de complexidade estão os problemas definíveis]por uma dada lógica. Nesta seção veremos como estabelecer esse tipo de relação e apresentaremos definições e resultados clássicos da área. Começaremos com uma breve revisão de lógica.

Um vocabulário relacional $\sigma = \langle R_1^{a_1}, \ldots, R_r^{a_r}, c_1, \ldots, c_s \rangle$ é uma tupla de símbolos relacionais R_i de aridade a_i e símbolos de constante. Uma estrutura $\mathcal{A}$ de vocabulário σ consiste de um conjunto não-vazio A, o domínio de $\mathcal{A}$, de uma relação a_i-ária $R_i^{\mathcal{A}}$ em A para todo símbolo relacional a_i-ário R_i em σ, e de um elemento $c_j^{\mathcal{A}}$ de A para toda constante em σ. Defina a linguagem de primeira ordem $\mathcal{L}[\sigma]$ como sendo o conjunto das fórmulas construídas utilizando os símbolos do vocabulário σ e os símbolos usuais da lógica de primeira ordem. Seja $\varphi \in \mathcal{L}[\sigma]$, denotamos por $\mathrm{Mod}(\varphi)$ o conjunto das estruturas de vocabulário σ que são modelos de φ. Para comparar o poder expressivo das lógicas introduziremos as seguintes relações:

Definição 2.2.1. Sejam $\mathcal{L}_1$ e $\mathcal{L}_2$ lógicas.

1. $\mathcal{L}_1 \leq \mathcal{L}_2$ se para todo σ e toda sentença $\varphi \in \mathcal{L}_1[\sigma]$ existir uma sentença $\psi \in \mathcal{L}_2[\sigma]$ tal que $\mathrm{Mod}(\varphi)$=$\mathrm{Mod}(\psi)$, i.e., $\mathcal{L}_2$ é pelo menos tão expressiva quanto $\mathcal{L}_1$.
2. $\mathcal{L}_1 < \mathcal{L}_2$ se $\mathcal{L}_1 \leq \mathcal{L}_2$ e não $\mathcal{L}_2 \leq \mathcal{L}_1$.
3. $\mathcal{L}_1 \equiv \mathcal{L}_2$ se $\mathcal{L}_1 \leq \mathcal{L}_2$ e $\mathcal{L}_2 \leq \mathcal{L}_1$, i.e., $\mathcal{L}_1$ e $\mathcal{L}_2$ tem o mesmo poder expressivo.

Um exemplo de comparação entre lógicas é FO $\leq$ SO, onde SO é a lógica de segunda ordem. Podemos também comparar a expressividade de lógicas quando impomos alguma restrição.

Definição 2.2.2. Chamamos de $\mathcal{L}^k$ a restrição da linguagem de primeira ordem onde somente as variáveis $x_1, \ldots, x_k$ ocorrem nas fórmulas.

Se considerarmos fórmulas que exprimem a cardinalidade do domínio de uma estrutura, facilmente verificamos que $\mathcal{L}^k < \mathcal{L}$. Denominaremos de FO^k o conjunto das consultas booleanas de primeira ordem expressas em $\mathcal{L}^k$.

Como nosso objeto de estudo se relaciona com a computação de uma máquina e os computadores são dispositivos finitos que lidam com objetos finitos, nos basearemos na teoria de modelos finitos e, portanto, todas as estruturas consideradas serão finitas. Chamaremos de STRUC[σ] o conjunto das estruturas finitas de vocabulário σ.

Além da restrição de ser finita, as estruturas deverão ter a relação de ordem. Mais precisamente, sempre que nos referirmos a um vocabulário σ, estaremos assumindo que $\sigma_0 \subseteq \sigma$, onde $\sigma_0 = \langle \leq, S, min, max \rangle$ e os símbolos $\leq$, S, min e max se referem à ordem total do domínio, à relação de sucessor e aos elementos menor e maior do domínio, respectivamente. Para uma estrutura de tamanho n, $|A| = n$, podemos assumir sem perdas

de generalidades que $A = \{0, 1, \ldots, n-1\}$ e que a relação $\leq$ é a ordem natural nesse conjunto. Tais restrições também são impostas pelo fato de estarmos lidando com computações, pois o conceito de ordenação está intimamente ligado a computação e a *strings* e, para produzir uma codificação não ambígua das estruturas, precisamos de ordem, como veremos mais adiante.

Como mencionado anteriormente, as consultas serão o nosso principal objeto de análise. Intuitivamente uma consulta em Complexidade Descritiva tem um sentido semelhante, porém mais geral, do que uma consulta em Banco de Dados. Existe uma linguagem, que será usada para definir a propriedade que a consulta deseja representar, e existem as estruturas relacionais, que são os bancos de dados. Quando aplicamos uma consulta a uma estrutura queremos saber se a estrutura tem a propriedade descrita na consulta ou queremos que ela retorne uma nova estrutura. Formalmente definimos consulta da seguinte forma:

Definição 2.2.3. Uma consulta é um mapeamento q : STRUC$[\sigma] \to$ STRUC$[\tau]$ do conjunto de estruturas do vocabulário *sigma* em um conjunto de estruturas do vocabulário τ. Uma consulta booleana é um mapeamento q_b : STRUC$[\sigma] \to \{0, 1\}$. Tais consultas podem ser vistas como um subconjunto de STRUC$[\sigma]$, o conjunto das estruturas $\mathcal{A}$ para as quais $q_b(\mathcal{A}) = 1$.

Para nossos propósitos, as consultas de primeira ordem são o tipo mais simples de consulta. Perceba que qualquer sentença de primeira ordem $\varphi \in \mathcal{L}[\sigma]$ define uma consulta booleana q_φ sobre STRUC$[\sigma]$ tal que $q_\varphi(\mathcal{A}) = 1$ sse $\mathcal{A} \models \varphi$. Denotaremos por FO e SO tanto as lógicas de primeira ordem e segunda ordem, respectivamente, como o conjunto das consultas booleanas definidas por sentenças dessas lógicas. Ficará claro pelo contexto qual o caso em questão. Quando quisermos nos referir ao fragmento existencial ou universal usaremos $\exists$ ou $\forall$, por exemplo, $\exists$SO é a lógica existencial de segunda ordem.

Exemplo 2.2.4. Considere o vocabulário de grafos $\tau_g = \langle E \rangle$ e a sentença em FO φ_{undir}=$\forall x \forall y(\neg E(x, x) \wedge (E(x, y) \to E(y, x)))$. Essa sentença define a consulta booleana I_{undir}:STRUC$[\tau_g] \to \{0, 1\}$ de forma que $\mathrm{I}_{undir}(\mathcal{A})$=1 sempre que $\mathcal{A} \in$ STRUC$[\tau_g]$ representar um grafo não direcionado e sem *loops*.

Como os problemas de decisão são o padrão para definir as classes de complexidades, as consultas booleanas, que claramente se relacionam com tais problemas, terão papel central em nosso estudo.

Como já sabemos, as classes de complexidade computacional são usualmente definidas como classes de linguagens decididas por máquinas de Turing. Essa forma de definir é bem natural, pois as máquinas de Turing computam funções entre *strings*. Em Complexidade Descritiva estamos interessados em relacionar consultas booleanas com máquinas de Turing e, como as consultas são funções sobre estruturas, precisamos representar as estruturas como *strings*.

Definição 2.2.5. Considere $\sigma = \langle R_1^{a_1}, \ldots, R_r^{a_r}, c_1, \ldots, c_s \rangle$ um vocabulário relacional e $\mathcal{A}$ uma estrutura ordenada de vocabulário σ e tamanho n. Codificaremos cada relação $R_i^{\mathcal{A}}$ como uma *string* binária $bin^{\mathcal{A}}(R_i)$ de tamanho n^{a_i} onde "1" na i-ésima posição indica que a i-ésima tupla, considerando a ordem lexicográfica, pertence à relação. Para as constantes, $bin^{\mathcal{A}}(c_j)$ será a codificação em binário do elemento a que $c_j^{\mathcal{A}}$ se refere. A *codificação binária da estrutura* $\mathcal{A}$ é a concatenação da codificação de cada relação e constante:

$$bin(\mathcal{A}) = bin^{\mathcal{A}}(R_1)bin^{\mathcal{A}}(R_2)\ldots bin^{\mathcal{A}}(c_s).$$

O tamanho da codificação da estrutura está intimamente ligado ao vocabulário e ao tamanho do seu domínio. Na codificação definida acima, uma relação k-ária em um domínio de tamanho n requer uma *string* de tamanho n^k para codificá-la. Com essa codificação temos uma forma de relacionar consultas booleanas com máquinas de Turing e, consequentemente, com classes de complexidade computacional.

Definição 2.2.6 (***q*** **está em** $\boldsymbol{\mathcal{C}}$**).** Considere q uma consulta booleana sobre um vocabulário σ e $\mathcal{C}$ uma classe de complexidade computacional. Dizemos que q *está em* $\mathcal{C}$ se a linguagem $\{bin(\mathcal{A}) \mid \mathcal{A} \in \text{STRUC}[\sigma] \text{ e } q(\mathcal{A}) = 1\}$ pertence à $\mathcal{C}$.

Agora que podemos relacionar consultas booleanas com classes de complexidade, podemos também relacionar a lógica utilizada para definir a consulta com as classes de complexidade. Dessa forma, usaremos FO para representar tanto a linguagem de primeira ordem quanto o conjunto das consultas booleanas definíveis na linguagem da lógica de primeira ordem. Vamos agora entender o que significa uma classe de complexidade computacional $\mathcal{C}$ ser capturada por uma lógica $\mathcal{L}$.

Definição 2.2.7 (**$\mathcal{L}$ está em $\mathcal{C}$**)**.** Considere $\mathcal{C}$ uma classe de complexidade computacional e $\mathcal{L}$ uma lógica. Dizemos que $\mathcal{L}$ *está em* $\mathcal{C}$ se, para todo vocabulário σ e toda sentença φ em $\mathcal{L}[\sigma]$, q_φ está em $\mathcal{C}$.

Definição 2.2.8 (**$\mathcal{L} = \mathcal{C}$**)**.** Dizemos que $\mathcal{L}$ *captura* $\mathcal{C}$ se o seguinte vale:

1. $\mathcal{L}$ está em $\mathcal{C}$.

2. Para toda consulta booleana q na classe de complexidade $\mathcal{C}$, existe uma sentença $\varphi_q \in \mathcal{L}$ tal que $\mathcal{A} \models \varphi_q$ sse $q(\mathcal{A}) = 1$.

Um dos primeiros resultados onde se demonstrou que uma lógica capturava uma classe de complexidade foi o teorema provado por Fagin em [Fag74] no qual ele enuncia que a lógica de segunda ordem existencial captura a classe NP, ∃SO=NP. Outro resultado importante diz respeito ao conjunto FO e à classe correspondente à hierarquia logarítmica, LH, o que nos mostra que o poder expressivo de FO é baixo. Apresentaremos um esboço da demonstração deste teorema para exibir uma das técnicas usadas para provar resultados desse tipo.

Teorema 2.2.9. *[Imm99]* FO = LH.

Demonstração. Para provar que FO ⊆ LH, para cada sentença $\varphi = \exists x_1 \forall x_2 \ldots Q_k x_k \alpha(\overline{x})$ em $\mathcal{L}_\sigma$, onde α é livre de quantificadores, que define uma consulta booleana q_φ :STRUC[σ]→ $\{0,1\}$, devemos construir uma máquina de Turing M que execute em tempo logarítmico tal que, para toda estrutura $\mathcal{A} \in$ STRUC[σ], $\mathcal{A}$ satisfaz φ sse M aceita a codificação binária de $\mathcal{A}$. Note que φ está descrita na forma normal prenex e como toda sentença tem uma equivalente na forma normal, estamos considerando todas as sentenças de FO. Em símbolos: $\forall \varphi, \mathcal{A} \models \varphi$ sse M($bin(\mathcal{A})$) ↓. Esta máquina é construída por indução em k. Para provar que LH ⊆ FO, devemos encontrar uma consulta booleana q que seja completa para LH via reduções de primeira ordem, mostrar que FO é fechada para reduções de primeira ordem e finalizar expressando q em FO. □

Na demonstração acima, para provar que LH ⊆ FO também podemos construir uma fórmula que expressa a computação de uma máquina em LH. Essa fórmula seria construída de acordo com a função de transição δ da máquina e teria como base o fato de uma máquina nessa classe executar em tempo logarítmico.

Além de resultados que relacionam lógicas com classes de complexidade, resultados relevantes na área de complexidade computacional foram obtidos a partir de resultados em complexidade descritiva. É o caso do teorema a seguir.

Teorema 2.2.10. *[Imm88][Sze88]* NSPACE($f(n)$)=coNSPACE($f(n)$)*, para toda função* $f(n) \geq \log n$.

Esse teorema respondeu uma questão que ficou em aberto por mais ou menos 25 anos e mostrou que o espaço não-determinístico é fechado para complemento, fato que era tido como falso. Na prova apresentada em [Imm88], o resultado segue como corolário dos seguintes teoremas:

Teorema 2.2.11. *[Imm88] Para estruturas finitas e ordenadas,* FO(pos TC) = FO(TC).

FO(TC) é a lógica de primeira ordem estendida com o operador de fecho transitivo. O operador de fecho transitivo define o fecho reflexivo e transitivo de uma relação binária definida por uma fórmula $\varphi(x, y)$. FO(pos TC) é a extensão onde todas as ocorrências do operador são positivas, ou seja, não estão no escopo de um número ímpar de negações.

Teorema 2.2.12. *[Imm88]* FO(pos TC) = NL.

Dessa forma, vemos que a área de complexidade descritiva, além de prover resultados relacionados à expressividade de lógicas, pode ajudar a solucionar problemas em complexidade computacional utilizando as técnicas e resultados da teoria de modelos finitos.

3 Lógica Modal

Neste capítulo investigaremos o poder expressivo de algumas lógicas modais e tentaremos expressar o problema REACH. Esse problema é importante, pois, como a execução de um algoritmo pode ser modelada através de um grafo que representa o espaço de busca do problema, a consulta booleana REACH, e suas variantes, são consideradas representativas de várias classes de complexidade [Pap94]. Dessa forma, se é possível expressar REACH com alguma lógica modal, talvez essa lógica capture a classe de complexidade para a qual REACH é completo.

Existem alguns resultados sobre que lógicas podem expreessar REACH. Como REACH fala da existência de um caminho entre dois vértices, para um dado número natural n, é possível definir, em FO, a noção de caminho de tamanho n. Porém, para um n qualquer, é preciso definir a noção de fecho transitivo e reflexivo de uma relação, o que não é possível usando FO [Lib04]. Assim, claramente podemos expressar REACH em FO(TC).

Na próxima seção apresentaremos algumas definições e teoremas importantes sobre a lógica modal básica encontrados em [BdRV01]. A seguir exploraremos seu poder expressivo e veremos que não é possível expressar REACH. Finalmente, investigaremos as lógicas CTL e CTL*. Veremos que elas são capazes de expressar REACH e mostraremos em que classes de complexidade elas estão. Os resultados deste capítulo estão publicados em [FM09].

3.1 Lógica Modal Básica

As linguagens modais são linguagens sintaticamente simples, porém são um instrumento poderoso para descrever propriedades de estruturas relacionais. Por causa de sua simplicidade, elas vêm se tornando bastante populares em diversas aplicações. Nesta seção, veremos algumas definições e teoremas importantes relacionados à lógica modal

básica.

Definição 3.1.1 (Linguagem Modal Básica)**.** A *linguagem modal básica* é definida por um conjunto Φ de letras proposicionais e um operador unário $\Diamond$ (diamante). O conjunto das fórmulas bem formadas (fbf) é o menor conjunto que satisfaz as seguintes condições:

1. Se $p \in \Phi$ então p é uma fbf.
2. $\bot$ é uma fbf.
3. Se φ é uma fbf então $(\neg\varphi)$ é uma fbf.
4. Se φ e ψ são fbf então $(\varphi \wedge \psi)$ é uma fbf.
5. Se φ é uma fbf então $(\Diamond\varphi)$ é uma fbf.

O operador $\Box$ (caixa) é o dual do $\Diamond$ (diamante), ou seja, $\Box\varphi \equiv \neg\Diamond\neg\varphi$. Abaixo, iremos definir a relação de satisfação entre uma fórmula modal e um modelo.

Definição 3.1.2 (Modelo)**.** Um *modelo* é uma tripla (W, R, V) tal que:

1. W é um conjunto não-vazio.
2. $R \subseteq W \times W$, é uma relação binária e representa a acessibilidade entre os elementos de W.
3. $V : \Phi \to \mathcal{P}$, é uma função de valoração que indica em que elementos de W uma letra proposicional assume valor verdadeiro.

Definição 3.1.3 (Satisfação)**.** Seja um modelo $M = (W, R, V)$ e $w \in W$. A relação de *satisfação* $\models$ é dada como segue:

1. $M, w \models p$ sse $w \in V(p)$, onde $p \in \Phi$.
2. $M, w \not\models \bot$.
3. $M, w \models (\neg\varphi)$ sse $M, w \not\models \varphi$.
4. $M, w \models (\varphi \wedge \psi)$ sse $M, w \models \varphi$ e $M, w \models \psi$.
5. $M, w \models (\Diamond\varphi)$ sse existe $v \in W$ tal que $R(w, v)$ e $M, v \models \varphi$.

Uma fórmula φ é *satisfatível* em um modelo M se existe $w \in M$ tal que $M, w \models \varphi$.

Definição 3.1.4. Uma fórmula φ é *globalmente satisfeita* em um modelo M, notação $M \models \varphi$, se $M, w \models \varphi$, para todo $w \in W$.

Definição 3.1.5 (Validade)**.** Uma fórmula φ é *válida*, $\models \varphi$, se $M \models \varphi$, para todo modelo M. Dizemos também que φ é *válida com relação a uma classe de modelos* $\mathfrak{M}$ se para todo $M \in \mathfrak{M}$, $M \models \varphi$.

A lógica modal básica não é um sistema isolado. Na verdade, é possível estabelecer relações entre ela e outras lógicas como, por exemplo, a lógica clássica de primeira ordem. Estabelecer relações desse tipo nos ajuda a analisar o poder expressivo da modalidade. A seguir definiremos o que chamamos de tradução padrão, que traduzirá fórmulas da linguagem modal básica em fórmulas da linguagem de primeira ordem.

Definição 3.1.6. Para um conjunto de letras proposicionais Φ, defina $\mathcal{L}^l(\Phi)$ como a linguagem de primeira ordem que tem predicados unários P_i correspondendo às letras proposicionais p_i em Φ e uma relação binária R.

Perceba que os modelos da linguagem modal básica podem ser vistos como estruturas relacionais $(W, R, P_0, P_1, \ldots)$ onde cada P_i é uma relação unária tal que $P_i(w) \leftrightarrow w \in V(p_i)$ e o símbolo R é interpretado como o R do modelo modal. Agora definiremos a tradução padrão.

Definição 3.1.7 (Tradução Padrão)**.** Considere x uma variável de primeira ordem. A *tradução padrão* ST_x que traduz fórmulas modais em fórmulas de primeira ordem de $\mathcal{L}^l(\Phi)$ é definida como segue:

1. $ST_x(p) = P(x)$.
2. $ST_x(\bot) = x \neq x$.
3. $ST_x((\neg\varphi)) = \neg ST_x(\varphi)$.
4. $ST_x((\varphi \wedge \psi)) = ST_x(\varphi) \wedge ST_x(\psi)$.
5. $ST_x((\Diamond\varphi)) = \exists y(Rxy \wedge ST_y(\varphi))$, onde y é uma variável que ainda não foi usada na tradução.

Como dito anteriormente, não há distinção matemática entre os modelos modais e de primeira ordem, pois os modelos modais podem ser facilmente transformados em uma estrutura relacional. Assim, podemos escrever $M \models ST_x(\varphi)[w]$ significando que a fórmula

de primeira ordem $ST_x(\varphi)$ é satisfeita no modelo M quando a variável x é substituída por w.

Teorema 3.1.8. *[BdRV01] Seja φ uma fórmula da linguagem modal básica. Temos que:*

1. *Para todo modelo M e todo w de M: $M, w \models \varphi$ sse $M \models ST_x(\varphi)[w]$.*
2. *Para todo modelo M: $M \models \varphi$ sse $M \models \forall x ST_x(\varphi)$.*

Teorema 3.1.9. *[BdRV01] Toda fórmula modal básica φ é equivalente a uma fórmula de primeira ordem com no máximo duas variáveis.*

Com os resultados acima podemos inferir que a Lógica Modal Básica não é mais expressiva que FO e, portanto, não é possível expressar REACH com ela.

3.2 As lógicas CTL* e CTL

Com os resultados da seção anterior concluímos que não é possível expressar REACH com a lógica modal básica. Dessa forma, investigaremos lógicas modais com poder expressivo maior, como é o caso de CTL e CTL*.

CTL* (*Computation Tree Logic*) é uma lógica temporal que pode ser usada para especificar propriedades de sistemas de transição de estados tais como os sistemas reativos. Tais sistemas estão continuamente interagindo com o ambiente podendo, inclusive, serem programados para não parar. CTL* permite modelar sequências de computação que serão representadas por árvores de computação obtidas a partir de uma estrutura de Kripke.

Definição 3.2.1 (Estrutura de Kripke)**.** Uma *estrutura de Kripke M* é um modelo conforme a Definição 3.1.2, no qual R é uma relação de transição total, ou seja, para qualquer $s \in W$ existe $s' \in W$ tal que Rss'. Chamaremos W de *conjunto de estados*.

Construímos a árvore de computação elegendo um estado para ser o inicial, a raiz, e obtendo os ramos infinitos da árvore que representam todos os caminhos possíveis a partir da raiz. As fórmulas de CTL* são de dois tipos: fórmulas de estado, que descrevem propriedades de um estado, e fórmulas de caminho, que descrevem propriedades de um caminho (veja Definição 3.2.3). CTL* é o conjunto das fórmulas de estado geradas pela seguinte sintaxe [CGP01]:

Definição 3.2.2 (Sintaxe de CTL*)**.** Seja Φ um conjunto de letras proposicionais, $\mathcal{S}$ o conjunto das fórmulas de estado e $\mathcal{P}$ o conjunto das fórmulas de caminho definidas indutivamente por:

1. Se $p \in \Phi$ então $p \in \mathcal{S}$.
2. Se φ e $\psi \in \mathcal{S}$ então $(\neg\varphi), (\varphi \wedge \psi) \in \mathcal{S}$.
3. Se $\varphi \in \mathcal{P}$ então $(E\varphi) \in \mathcal{S}$.
4. Se $\varphi \in \mathcal{S}$ então $\varphi \in \mathcal{P}$.
5. Se φ e $\psi \in \mathcal{P}$ então $(\neg\varphi), (\varphi \wedge \psi), (X\varphi), (\varphi U \psi) \in \mathcal{P}$.

Na definição 3.2.2, temos operadores temporais (modais) e um quantificador de caminho. Os operadores temporais são: $X(next)$, e $X\varphi$ intuitivamente significa que φ é verdade no próximo estado, e $U(until)$, tal que $\varphi U \psi$ significa que existe um estado no qual ψ é verdade e, do estado atual até este estado, φ é verdade. O quantificador de caminho E é usado sobre uma fórmula de caminho φ e $E\varphi$ significa que existe um caminho a partir de certo estado tal que φ é satisfeito naquele caminho. Agora precisamos definir a semântica de CTL* com respeito a uma estrutura de Kripke e, para isto, precisamos definir o que é um caminho nessa estrutura.

Definição 3.2.3 (Caminho em uma estrutura de Kripke)**.** Um *caminho em uma estrutura de Kripke* M é uma sequência infinita de estados $\pi = s_0 s_1 s_2 ...$ tal que, para todo $i \geq 0$, $Rs_i s_{i+1}$. Denotamos por π^i o sufixo de π começando em s_i.

Se $\varphi \in \mathcal{S}$ usamos a notação $M, s \models \varphi$, que significa que φ vale no estado s do modelo M, e se $\varphi \in \mathcal{P}$ usamos $M, \pi \models \varphi$, que significa que φ vale no caminho π do modelo M. A relação $\models$ é definida indutivamente a seguir.

Definição 3.2.4. Assuma que $\varphi_1, \varphi_2 \in \mathcal{S}$ e $\psi_1, \psi_2 \in \mathcal{P}$. A relação de *satisfação* é definida como segue:

1. $M, s \models p$ *sse* $p \in \Phi$.
2. $M, s \models \neg\varphi_1$ *sse* $M, s \not\models \varphi_1$.
3. $M, s \models \varphi_1 \wedge \varphi_2$ *sse* $M, s \models \varphi_1$ e $M, s \models \varphi_2$.
4. $M, s \models E\psi_1$ *sse* existe π partindo de s tal que $M, \pi \models \psi_1$.
5. $M, \pi \models \varphi_1$ *sse* $\pi = s\pi^1$ e $M, s \models \varphi_1$.
6. $M, \pi \models \neg\psi_1$ *sse* $M, \pi \not\models \psi_1$.
7. $M, \pi \models \psi_1 \wedge \psi_2$ *sse* $M, \pi \models \psi_1$ e $M, \pi \models \psi_2$.
8. $M, \pi \models X\psi_1$ *sse* $M, \pi^1 \models \psi_1$.
9. $M, \pi \models \psi_1 U \psi_2$ *sse* existe $k \geq 0$ tal que $M, \pi^k \models \psi_2$ e, para todo $0 \leq j \leq k, M, \pi^j \models \psi_1$.

CTL é a sublinguagem de CTL* na qual toda ocorrência de um operador temporal é precedida imediatamente por um quantificador de caminho. Alguns resultados desta lógica serão exibidos a seguir.

Teorema 3.2.5. *[Imm99] Toda fórmula em* CTL *pode ser traduzida para uma fórmula equivalente em* $\mathrm{FO}^2(\mathrm{TC})$.

Sabendo que REACH pode ser expresso em FO(TC), iremos mostrar que REACH pode ser expresso em CTL.

Teorema 3.2.6. REACH *pode ser expresso em* CTL *e* CTL*.

Demonstração. REACH é o problema de decidir se existe um caminho entre dois vértices específicos de um grafo direcionado. Seja M uma estrutura de Kripke modificada tal que $M = (W, R, R_s, R_t, V)$, onde W representa o conjunto de vértices do grafo, R é a relação binária que representa as arestas entre os vértices, R_s e R_t são relações unárias que dizem quem é s e t respectivamente e V é a função de valoração. R_s e R_t definem dois operadores modais 0-ários, $\Diamond_s$ e $\Diamond_t$, tais que $M, w \models \Diamond_s$ sse $w \in R_s$, ou seja, w é s e $M, w \models \Diamond_t$ sse $w \in R_t$, ou seja, w é t. REACH pode ser definido com a fórmula $EF(\Diamond_s \wedge EF\Diamond_t)$, onde $F\varphi \equiv trueU\varphi$. Pela Definição 3.2.4, $M \in$ REACH sse $M, w_0 \models EF(\Diamond_s \wedge EF\Diamond_t)$, onde w_0 é a raiz. Logo REACH pode ser expresso em CTL, e em CTL*, já que CTL é um fragmento de CTL*. □

Teorema 3.2.7. CTL *e* CTL* *está em* NL *(Espaço logarítmico não-determinístico) e não está em* LH.

Demonstração. Como NL=FO(TC) [Imm99] e $\mathrm{FO}^2(\mathrm{TC})$ é menos expressivo que FO(TC), segue direto do Teorema 3.2.5 que CTL está em NL. Porém, como FO é menos expressivo que FO(TC), pelo Teorema 2.2.9 temos que CTL não está em LH. □

4 Lógicas com Operadores de Ponto Fixo

A lógica clássica de primeira-ordem já foi bastante estudada no escopo de Complexidade Descritiva e sabemos que ela não é muito expressiva [Imm99]. Pensando em como estender essa lógica vemos várias possibilidades que podem aumentar o seu poder expressivo, dentre elas: utilizar variáveis de ordem mais alta, permitir que as fórmulas tenham tamanho infinito ou adicionar à lógica operadores que ela não é capaz de definir. Neste capítulo vamos investigar a adição de operadores de ponto fixo à FO. Iniciaremos com uma breve revisão do conceito de ponto fixo e depois analisaremos como a adição desses operadores pode aumentar a expressividade de FO.

4.1 Operadores de Ponto Fixo

No texto que segue, considere um conjunto finito M qualquer e uma função total $F : \mathcal{P}(M) \longrightarrow \mathcal{P}(M)$ que mapeia o conjunto das partes de M nele mesmo. Quando aplicamos F a um conjunto $X \subseteq M$ e obtemos como imagem o próprio conjunto X, dizemos que X é um ponto fixo de F. Formalmente,

Definição 4.1.1 (Ponto Fixo)**.** Se existir $X \subseteq M$ tal que $F(X) = X$ dizemos que X é um *ponto fixo* de F. Além disso, se X estiver contido em todo ponto fixo de F dizemos que X é o menor ponto fixo de F.

Exemplo 4.1.2. Pela definição podemos ver que uma função pode ter vários pontos fixos ou mesmo nenhum. A função identidade, $I(X) = X$, tem vários pontos fixos, pois todo subconjunto de M é ponto fixo de I, enquanto que a função complemento, $C(X) = M - X$, não tem nenhum ponto fixo.

Apesar de nem toda função possuir ponto fixo, existem algumas propriedades que garantem a existência de pelo menos 1 ponto fixo. Considere a sequência $\emptyset$, $F(\emptyset)$, $F(F(\emptyset))$, ..., definida indutivamente da seguinte forma:

$$\begin{cases} F_0 = \emptyset, \\ F_{n+1} = F(F_n). \end{cases}$$

Dizemos que F_n é o n-ésimo estágio de F e, se houver um n_0 tal que $F(F_{n_0}) = F_{n_0}$, então F_{n_0} é um ponto fixo de F. Observe que, para todo $n \geq n_0$, $F_n = F_{n_0}$ e, portanto, a sequência $F_0, F_1, F_2, \ldots$ convergiu. Nesse caso, denotamos por F_∞ o valor de convergência da sequência, que é um ponto fixo de F, ou seja, $F_\infty = F_{n_0}$. Uma função pode ter algum ponto fixo e não ter o ponto fixo F_∞, porém podemos garantir que F_∞ existe quando a função F é indutiva.

Definição 4.1.3. Dizemos que F é *indutiva* se $F_0 = \emptyset$ e $F_i \subseteq F_{i+1}$, para todo i.

Exemplo 4.1.4. Considere $M = \{1, 2, \ldots, n\}$. A função $G : \mathcal{P}(M) \to \mathcal{P}(M)$ é definida como $G(X) = X \cup \{|X| + 1\}$, quando $X \neq M$, e $G(X) = X$, quando $X = M$. Nesse caso, $G_0 = \emptyset \subseteq G_1 = \{1\} \subseteq G_2 = \{1, 2\} \ldots$ e claramente G é indutiva.

Lema 4.1.5. *Considere $F : \mathcal{P}(M) \longrightarrow \mathcal{P}(M)$ e $|M| = m$.*

1. *Se F_∞ existe, então $F_\infty = F_{2^m - 1}$.*
2. *Se F é indutivo, então F_∞ existe e $F_\infty = F_m$.*

Demonstração.

1. Suponha que F_∞ existe. Como $\mathcal{P}(M)$ tem 2^m elementos, existem $n < 2^m$ e $l \geq 1$ tais que $F_n = F_{n+l}$. Se $l = 1$ então $F_n = F_{n+1}$, e portanto $F_n = F_{2^m - 1} = F_\infty$. Se $F_n \neq F_{n+1}$ então para $k \geq 0$ temos $F_{n+k.l} \neq F_{n+k.l+1}$ e portanto F_∞ não existe.
2. Por definição, $F_0 \subseteq F_1 \subseteq \ldots \subseteq M$. Como M tem m elementos, a sequência fica constante no máximo no estágio F_m.

□

No primeiro caso basta notar que, se o ponto fixo F_∞ existe, ele será alcançado em algum estágio F_n. Como só podem existir no máximo 2^m estágios diferentes então ele será alcançado no estágio $F_{2^m - 1}$. No segundo caso, como a função é indutiva, podemos afirmar que a cada estágio pelo menos um elemento é adicionado e, dessa forma, o ponto fixo F_∞ é atingido no máximo no estágio m e, nesse caso, o ponto fixo será o próprio M.

Como vimos no Lema 4.1.5, a propriedade ser indutiva garante a existência do ponto fixo F_∞ e, além disso, mostra que esse ponto fixo pode ser alcançado em um número

exponencialmente menor de estágios do que quando consideramos o ponto fixo de uma função qualquer. Dessa forma, calcular o ponto fixo de uma função indutiva é bem mais fácil, computacionalmente falando, do que calculá-lo para uma função qualquer e por isso é interessante investigar em que condições uma função é indutiva. A seguir veremos duas propriedades que garantem que a função seja indutiva.

Definição 4.1.6. Dizemos que F é *inflacionária* se $X \subseteq F(X)$, para todo $X \subseteq M$.

Definição 4.1.7. Dizemos que F é *monótona* se $F(R) \subseteq F(S)$, para todo $R \subseteq S \subseteq M$.

Lema 4.1.8. *Considere $F : \mathcal{P}(M) \longrightarrow \mathcal{P}(M)$ e $|M| = m$.*

1. *Se F é inflacionária ou monótona então F é indutiva.*
2. *Se F é monótona então F_∞ é o menor ponto fixo de F.*
3. *Considere F uma função qualquer e $F' : \mathcal{P}(M) \longrightarrow \mathcal{P}(M)$ tal que $F'(X) = X \cup F(X)$ então F' é inflacionária. Se F for indutivo então $F'_n = F_n$, para todo $n \geq 0$, e $F'_\infty = F_\infty$.*

Demonstração.

1. Se F é inflacionária então $F_n \subseteq F(F_n) = F_{n+1}$, portanto F é indutiva. Se F é monótona temos que $F_0 = \emptyset \subseteq F(\emptyset) = F_1$ e, pela motonicidade, $F_1 = F(F_0) \subseteq F(F_1) = F_2$. Prosseguindo dessa forma, obtemos que $F_n \subseteq F_{n+1}$, para todo n e, assim, F é indutiva.

2. Seja X um ponto fixo qualquer de F. Como $\emptyset \subseteq X$ temos, por monotonicidade, que $F_1 = F(F_0) \subseteq F(X) = X$. Suponha que $F_n \subseteq X$. Logo, $F_{n+1} = F(F_n) \subseteq F(X) = X$. Assim, $F_\infty \subseteq X$ e, portanto, F_∞ é o menor ponto fixo de F.

3. Pela definição de F', $X \subseteq F'(X)$ para todo $X \subseteq M$. Logo, F' é inflacionária. Claramente, $F_0 = \emptyset = F'_0$. Suponha que $F_{n-1} = F'_{n-1}$. Temos que $F_n = F(F_{n-1})$ e $F'_n = F'(F'_{n-1}) = F'(F_{n-1}) = F_{n-1} \cup F(F_{n-1}) = F_{n-1} \cup F_n$. Como F é indutiva então $F_{n-1} \subseteq F_n$ e $F_{n-1} \cup F_n = F_n$. Assim, $F'_n = F_n$ para todo $n \geq 0$ e $F'_\infty = F_\infty$.

□

Na próxima seção veremos como adicionar operadores de ponto fixo à lógica e assim aumentar seu poder expressivo. Os resultados que analisam o ganho de expressividade estão intimamente relacionados com o lemas apresentados nessa seção.

4.2 Lógicas de Ponto Fixo

Como visto no Capítulos 3, FO não consegue expressar a consulta REACH. Entretanto, se quisermos expressar a consulta "existe um caminho de tamanho n entre s e t no grafo G", podemos fazê-lo usando FO.

Pensando em termos computacionais, FO tem o poder de expressar laços limitados (*for*), mas quando precisamos de um laço ilimitado (*while*) ela não é suficiente. Nesta seção veremos que ao adicionar operadores de ponto fixo apropriados à FO ganhamos o poder de expressar o laço ilimitado, mais precisamente, poderemos definir relações indutivas e relações iterativas.

Para explicar essa noção de definir uma relação indutiva considere um vocabulário relacional σ e uma fórmula $\varphi(x, y, R)$, onde x e y são as únicas variáveis livres de φ e R é um símbolo relacional binário que não está em σ. Para uma estrutura $\mathcal{A}$ desse vocabulário, φ induz uma função $F^{\varphi} : \mathcal{P}(A^2) \longrightarrow \mathcal{P}(A^2)$ tal que

$$F^{\varphi}(R) = \{(a, b) \mid \mathcal{A} \models \varphi(a, b, R)\}.$$

Da mesma forma que na seção anterior, definimos $F^{\varphi}_0 = \emptyset$ e $F^{\varphi}_{n+1} = F^{\varphi}(F^{\varphi}_n)$ e o ponto fixo F^{φ}_{∞}, se ele existir. Pensando apenas na função F^{φ} sabemos o que é necessário para que F^{φ}_{∞} exista. Como veremos no lema seguinte, restrições sintáticas em φ fazem com que F^{φ} seja monótona e, portanto, indutiva.

Lema 4.2.1. *Considere R um novo símbolo relaçional de aridade k e $\varphi(x_1, \ldots, x_k, R)$ uma fórmula. Se R ocorre positivamente em φ então F^{φ} é monótona.*

Dizer que R ocorre positivamente ou é positivo em φ quer dizer que R não ocorre no escopo de um número ímpar de negações. A prova do lema acima é feita por indução nas fórmulas, mas a intuição não é complicada. Queremos provar que se $X \subseteq Y \subseteq A^k$ então $F^{\varphi}(X) \subseteq F^{\varphi}(Y)$. Para esse fato não ser verdade teríamos que ter uma tupla em $F^{\varphi}(X)$ que não está em $F^{\varphi}(Y)$. Isso não é possível, pois, como $X \subseteq Y$, a única forma dessa tupla não estar em $F^{\varphi}(Y)$ seria se na fórmula φ tivesse alguma restrição negativa com relação a R, o que não ocorre.

Sabendo que quando R ocorre positivo em φ temos F^{φ} monótona, podemos afirmar que F^{φ}_{∞} existe e é um ponto fixo de F^{φ}. De fato, F^{φ}_{∞} é uma nova relação no domínio A. Para esclarecer como podemos utilizar esse fato, segue um exemplo.

Exemplo 4.2.2. Considere o vocabulário de grafos $\tau_g = \langle E \rangle$. Podemos definir o fecho transitivo e reflexivo da relação E de uma estrutura $\mathcal{G}$ como sendo $F_\infty^{\varphi_t}$, onde

$$\varphi_t(x, y, R) \equiv (x = y) \vee \exists z(E(x, z) \wedge R(z, y))$$

Mais claramente, considere $F_0^{\varphi_t} = \emptyset$. Assim temos:

$$F_1^{\varphi_t} = \{(a, b) \in G^2 \mid d(a, b) \leq 0\}$$

$$F_2^{\varphi_t} = \{(a, b) \in G^2 \mid d(a, b) \leq 1\}$$

$$\vdots$$

$$F_r^{\varphi_t} = \{(a, b) \in G^2 \mid d(a, b) \leq r - 1\},$$

onde $d(a, b)$ é a distância entre os vértices a e b em $\mathcal{G}$.

Como R ocorre positivamente em φ_t temos que F^{φ_t} é monótona e, pelo lema 4.1.8, $F_\infty^{\varphi_t}$ é o menor ponto fixo de F^{φ_t}. Claramente esse ponto fixo é o fecho transitivo e reflexivo da relação E. Em particular, se $|G| = n$ temos $F_n^{\varphi_t} = F_\infty^{\varphi_t}$.

Introduziremos, a seguir, a definição da lógica de menor ponto fixo FO(LFP).

Definição 4.2.3 (FO(LFP))**.** Considere σ um vocabulário relacional, R uma variável relacional de aridade k, $\varphi(R, x_1, \ldots, x_k)$ uma fórmula de FO(LFP) e $\mathcal{A} \in \text{STRUC}[\sigma]$. A linguagem de FO(LFP) estende a linguagem de FO com a seguinte regra de formação:

- se $\varphi(x_1, \ldots, x_k, R)$ é positiva em R e $(x_1, \ldots, x_k)$ é uma tupla de variáveis, então $[\mathbf{lfp}_{R,x_1,\ldots,x_k}\varphi(R, x_1, \ldots, x_k)]$ é uma fórmula na qual as variáveis livres são $(x_1, \ldots, x_k)$.

A relação de satisfatibilidade de FO(LFP) estende a de FO com a seguinte definição:

- $\mathcal{A} \models [\mathbf{lfp}_{R,x_1,\ldots,x_k}\varphi(R, x_1, \ldots, x_k)](a_1, \ldots, a_k)$ sse $(a_1, \ldots, a_k) \in F_\infty^{\varphi}$.

Com FO(LFP) podemos expressar a consulta REACH da seguinte forma:

$$\text{REACH} \equiv [\mathbf{lfp}_{R,x,y}\varphi_t](s, t),$$

onde φ_t é como definido no Exemplo 4.2.2.

Como visto no Teorema 2.2.9, FO captura a hierarquia logarítmica. Ao adicionar o operador de menor ponto fixo, somos capazes de definir relações indutivas e assim

temos uma lógica mais expressiva que FO. Naturalmente, tal lógica captura uma classe de complexidade mais poderosa que LH.

Teorema 4.2.4. *[Imm99] Para estruturas finitas e ordenadas,* FO(LFP) = P.

Na prova do teorema acima usa-se o fato de FO=LH e de REACH_a, uma variante do problema REACH, ser completo para a classe P via reduções de primeira-ordem. Definimos REACH_a como sendo o conjunto de grafos para os quais existem um caminho alternante entre s e t.

Para mostrar que FO(LFP) está em P devemos construir uma máquina em P para cada sentença em FO(LFP) tal que cada máquina compute a consulta definida por uma sentença. O trabalho da máquina será, essencialmente, calcular a nova relação definida pelo operador de menor ponto fixo e, como já foi visto, isso custará no máximo n^k passos, onde n é o tamanho do domínio e k a aridade da nova relação. Após isso, a máquina deve apenas verificar se a nova estrutura satisfaz a fórmula. Como isso demanda tempo logarítmico, o que prevalece é o tempo gasto para calcular a nova relação e, portanto, a máquina está em P. A outra direção é bem simples, pois como REACH_a é completo para P todo problema nessa classe pode ser reduzido a ele. Podemos expressar REACH_a com FO(LFP) com a seguinte fórmula $[\mathbf{lfp}_{R,x_1,x_2}\varphi_a(R,x_1,x_2)](s,t)$, onde $\varphi_a(R,x,y) \equiv (x = y) \vee (\exists z(E(x,z) \wedge R(z,y)) \wedge (A(x) \rightarrow \forall z(E(x,z) \rightarrow R(z,y))))$, onde A é o conjunto dos vértices que são universais. Assim, todo problema da classe P pode ser expresso por uma fórmula em FO(LFP).

Alguns detalhes técnicos foram omitidos nessa explicação, porém ela reflete a idéia central da demonstração e, mais ainda, reflete a idéia geral de como provas de igualdade entre classes de complexidade e lógicas podem ser feitas.

Devido ao resultado provado por Fagin, NP=∃SO, uma observação é que agora temos um problema equivalente ao P=NP em lógica, provar se FO(LFP)=∃SO. Vemos também que podemos definir relações indutivas com FO(LFP), porém precisamos atentar para a restrição sintática de que R ocorra positivamente em φ. Pensando que essa restrição pode eventualmente dificultar a expressão de alguma consulta, definiremos a lógica de ponto fixo inflacionária. Nessa lógica não são impostas restrições sintáticas à φ, mas é possível definir relações indutivas.

Lema 4.2.5. *Seja R um novo símbolo relacional de aridade k e $\varphi(R,x_1,\ldots,x_k)$ uma fórmula. Se $\varphi'(x_1,\ldots,x_k,R) = \varphi(R,x_1,\ldots,x_k) \vee R(x_1,\ldots,x_k)$ então $F^{\varphi'}$ é uma função inflacionária.*

O lema acima tem relação direta com o lema 4.6 e é esse fato que usaremos ao definir FO(IFP).

Definição 4.2.6 (FO(IFP))**.** Considere σ um vocabulário relacional, R uma variável relacional de aridade k, $\varphi(R, x_1, \ldots, x_k)$ uma fórmula de FO(IFP) e $\mathcal{A} \in \text{STRUC}[\sigma]$. A linguagem de FO(IFP) estende a linguagem de FO com a seguinte regra de formação:

- se $\varphi(x_1, \ldots, x_k, R)$ é uma fórmula e $(x_1, \ldots, x_k)$ é uma tupla de variáveis, então $[\mathbf{ifp}_{R,x_1,\ldots,x_k}\varphi(R, x_1, \ldots, x_k)]$ é uma fórmula na qual as variáveis livres são $(x_1, \ldots, x_k)$.

A relação de satisfatibilidade de FO(IFP) estende a de FO com a seguinte definição:

- $\mathcal{A} \models [\mathbf{ifp}_{R,x_1,\ldots,x_k}\varphi(R, x_1, \ldots, x_k)](a_1, \ldots, a_k)$ sse $(a_1, \ldots, a_k) \in F^{\varphi'}_{\infty}$, onde $\varphi'(x_1, \ldots, x_k, R) = \varphi(x_1, \ldots, x_k, R) \vee R(x_1, \ldots, x_k)$.

Veja que a função $F^{\varphi'}$ pode não ser monótona e, portanto, $F^{\varphi'}_{\infty}$ pode não ser o menor ponto fixo. Isso não é um problema, pois o que queremos garantir é que $F^{\varphi'}$ seja indutiva.

No lema 4.6 vimos que se uma função é inflacionária ou monótona então ela é indutiva e com o lema 4.3 garantimos que o ponto fixo F_{∞} sempre existe e é atingido em um número polinomial de passos. Estes resultados em conjunto com as definições das lógicas demonstram o seguinte teorema.

Teorema 4.2.7. *[EF95]* FO(IFP) = FO(LFP).

FO(IFP) e FO(LFP) têm o mesmo poder expressivo, porém em geral é mais simples definir consultas utilizando FO(IFP) porque não precisamos nos preocupar com as restrições sintáticas que são impostas no caso de FO(LFP).

As relações indutivas são um caso particular de relação iterativa. Se quisermos definir relações iterativas não precisaremos impor restrições, como é o caso das lógicas anteriores, apenas definir a noção de laço. Vejamos a definição da lógica de ponto fixo parcial.

Definição 4.2.8 (FO(PFP))**.** Considere σ um vocabulário relacional, R uma variável relacional de aridade k, $\varphi(R, x_1, \ldots, x_k)$ uma fórmula de FO(PFP) e $\mathcal{A} \in \text{STRUC}[\sigma]$. A linguagem de FO(PFP) estende a linguagem de FO com a seguinte regra de formação:

- se $\varphi(x_1, \ldots, x_k, R)$ é uma fórmula e $(x_1, \ldots, x_k)$ é uma tupla de variáveis, então $[\mathbf{pfp}_{R,x_1,\ldots,x_k}\varphi(R, x_1, \ldots, x_k)]$ é uma fórmula na qual as variáveis livres são $(x_1, \ldots, x_k)$.

A relação de satisfatibilidade de FO(PFP) estende a de FO com a seguinte definição:

- $\mathcal{A} \models [\mathbf{pfp}_{R,x_1,\ldots,x_k}\varphi(R,x_1,\ldots,x_k)](a_1,\ldots,a_k)$ sse $(a_1,\ldots,a_k) \in F^{\varphi}_{\infty}$, e caso F^{φ}_{∞} não exista, consideramos $F^{\varphi}_{\infty} = \emptyset$.

Na definição de FO(PFP) não impomos restrição alguma à φ e, dessa forma, não podemos garantir nenhuma das propriedades, como ser monótona ou inflacionária, para F^{φ}. Como visto no lema 4.3 não podemos garantir que existe o ponto fixo F^{φ}_{∞} para F^{φ} e, mais ainda, se ele existir pode ser atingido só após um número exponencial de passos. Dado isso, temos o seguinte resultado.

Teorema 4.2.9. *[Imm99] Para estruturas finitas e ordenadas,* FO(PFP) = PSPACE.

De forma simplificada, na prova desse teorema, fixamos que o número máximo de passos é exponencial, pelo lema 4.3, e utilizamos espaço polinomial para fazer essa contagem. Sabemos que P $\subseteq$ PSPACE, mas não podemos afirmar que P $\neq$ PSPACE. Assim podemos afirmar que:

Teorema 4.2.10. *[EF95]* FO(LFP) $\leq$ FO(PFP).

Novamente encontramos um equivalente na lógica para um problema de igualdade entre classes de complexidade computacional, pois se provarmos que FO(LFP) é menos expressiva que FO(PFP) então temos P $\subset$ PSPACE.

5 Lógicas de Ordem Superior

Como citado no Capítulo 4, existem várias formas de estender FO para obter lógicas com poder expressivo maior. Neste capítulo faremos uma revisão sobre as lógicas de ordem superior (HO^i), que são extensões de FO que nos permitem quantificar variáveis de qualquer ordem. Apresentaremos sua definição e alguns resultados com respeito a seu poder expressivo, que é maior que o de FO.

5.1 Sintaxe e Semântica

Para todo $i \geq 2$, o alfabeto da lógica de ordem superior de ordem i (HO^i) possui os símbolos lógicos e de pontuação usuais, um conjunto infinito contável de variáveis individuais e, para toda aridade $r \geq 1$ e toda ordem $2 \leq j \leq i$, um conjunto infinito contável de variáveis relacionais. Usaremos letras minúsculas como x, y para variáveis individuais e letras maiúsculas como X, Y para variáveis relacionais.

Definição 5.1.1. Seja σ um vocabulário relacional. Definimos indutivamente o *conjunto das fórmulas bem formadas (fbf) de* HO^i como segue:

1. Se x_1 e x_2 são variáveis individuais, então $x_1 = x_2$ é uma fbf.
2. Se R é um símbolo relacional em σ de aridade r e $x_1, \ldots, x_r$ são variáveis individuais, então $R(x_1, \ldots, x_r)$ é uma fbf.
3. Se X é uma variável relacional de ordem 2 e de aridade r e $x_1, \ldots, x_r$ são variáveis individuais, então $X(x_1, \ldots, x_r)$ é uma fbf.
4. Se X é uma variável relacional de ordem j, para $3 \leq j \leq i$, e de aridade r e $Y_1, \ldots, Y_r$ são variáveis relacionais de ordem $j-1$ e aridade r, então $X(Y_1, \ldots, Y_r)$ é uma fbf.
5. Se X_1 e X_2 são variáveis relacionais de ordem j, $2 \leq j \leq i$, e mesma aridade, então $X_1 = X_2$ é uma fbf.

6. Se φ e ψ são fbf, então $(\neg\varphi)$, $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$ são fbf's.

7. Se φ é uma fbf e x é uma variável individual, então $\exists x\varphi$ e $\forall x\varphi$ são fbf's.

8. Se φ é uma fbf e X é uma variável relacional, então $\exists X\varphi$ e $\forall X\varphi$ são fbf's.

Sem perda de generalidades assumimos que os elementos de uma relação de ordem i e aridade r são tuplas compostas por relações de ordem $i-1$ e aridade r, pois tal restrição simplifica as definições e a demonstração de alguns resultados. As fórmulas atômicas são as descritas nos itens 1 a 5 e somente as variáveis individuais podem ocorrer livres nas fórmulas.

Seja σ um vocabulário relacional. Uma *valoração* val em uma σ-estrutura $\mathcal{A}$, com domínio A, é uma função que atribui a cada variável individual x um elemento em A e a cada variável relacional X de aridade $r \geq 1$ e ordem j, $2 \leq j \leq i$, uma relação de ordem j e aridade r em A. Sejam val_0 e val_1 duas valorações em uma σ-estrutura $\mathcal{A}$ e seja V uma variável de qualquer tipo. Dizemos que val_0 e val_1 são *V-equivalentes* se elas coincidem em todas as variáveis de qualquer tipo, podendo diferir apenas na variável V.

Definição 5.1.2. Seja $\mathcal{A}$ uma σ-estrutura e val uma valoração em $\mathcal{A}$. A noção de *satisfação* em HO^i é definida como segue:

1. $\mathcal{A}, val \models R(x_1, \ldots, x_r)$:sse $(val(x_1), \ldots, val(x_r)) \in R^{\mathcal{A}}$, onde R é um símbolo relacional em σ de aridade r e $x_1, \ldots, x_r$ são variáveis individuais.

2. $\mathcal{A}, val \models X(x_1, \ldots, x_r)$:sse $(val(x_1), \ldots, val(x_r)) \in val(X)$, onde X é uma variável relacional de ordem 2 e de aridade r e $x_1, \ldots, x_r$ são variáveis individuais.

3. $\mathcal{A}, val \models X(Y_1, \ldots, Y_r)$:sse $(val(Y_1), \ldots, val(Y_r)) \in val(X)$, onde X é uma variável relacional de ordem j, $3 \leq j \leq i$, de aridade r e $Y_1, \ldots, Y_r$ são variáveis relacionais de ordem $j-1$ e aridade r.

4. $\mathcal{A}, val \models (x_1 = x_2)$:sse $val(x_1) = val(x_2)$, onde x_1 e x_2 são variáveis individuais.

5. $\mathcal{A}, val \models (X_1 = X_2)$:sse $val(X_1) = val(X_2)$, onde X_1 e X_2 são variáveis de mesma ordem e mesma aridade.

6. $\mathcal{A}, val \models (\neg\varphi)$:sse $\mathcal{A}, val \not\models \varphi$, onde φ é uma fbf.

7. $\mathcal{A}, val \models (\varphi \wedge \psi)$:sse $\mathcal{A}, val \models \varphi$ e $\mathcal{A}, val \models \psi$, onde φ e ψ são fbf.

8. $\mathcal{A}, val \models (\varphi \vee \psi)$:sse $\mathcal{A}, val \models \varphi$ ou $\mathcal{A}, val \models \psi$, onde φ e ψ são fbf.

9. $\mathcal{A}, val \models \exists x\varphi$:sse existe uma valoração val' que é x-equivalente a val e $\mathcal{A}, val' \models \varphi$, onde x é uma variável individual e φ é uma fbf.

10. $\mathcal{A}, val \models \forall x\varphi$:sse para toda valoração val' que é x-equivalente a val temos $\mathcal{A}, val' \models \varphi$, onde x é uma variável individual e φ é uma fbf.

11. $\mathcal{A}, val \models \exists X\varphi$:sse existe uma valoração val' que é X-equivalente a val e $\mathcal{A}, val' \models \varphi$, onde X é uma variável relacional e φ é uma fbf.

12. $\mathcal{A}, val \models \forall X\varphi$:sse para toda valoração val' que é X-equivalente a val temos $\mathcal{A}, val' \models \varphi$, onde X é uma variável relacional e φ é uma fbf.

Como já mencionado, em HO^i podemos utilizar variáveis de diversas ordens. As variáveis individuais designam algum elemento do domínio da estrutura e as variáveis relacionais designam subconjuntos desse domínio. A noção de relação de ordem i é dada como segue: Seja R uma relação de ordem i e aridade r. Se $i = 2$, então $R \in \mathcal{P}(A^r)$, ou seja, R é uma relação em A. Quando $i \geq 3$ temos que $R \in \mathcal{P}(A^r_{i-1})$, onde A_{i-1} é o conjunto das relações de ordem $i-1$ em A, ou seja, R é uma relação sobre o conjunto de relações de ordem $i-1$.

A lógica de segunda-ordem, HO^2, já foi bastante estudada e existem diversos exemplos de propriedades que podem ser expressas utilizando-a. No exemplo a seguir temos uma sentença que expressa a propriedade de um grafo poder ser colorido com 3 cores.

Exemplo 5.1.3. O problema de 3-coloração em grafos é definido como: Dado um grafo $\mathcal{G} = (V, E)$, existe uma coloração dos vértices com 3 cores tal que não há dois vértices adjacentes com a mesma cor? Podemos expressar tal propriedade com a seguinte sentença de lógica existencial de segunda-ordem:

$$\phi_{3color} \equiv \exists R \exists Y \exists B \forall x((R(x) \vee Y(x) \vee B(x)) \wedge (\forall y(E(x,y) \rightarrow$$

$$\neg(R(x) \wedge R(y)) \wedge \neg(Y(x) \wedge Y(y)) \wedge \neg(B(x) \wedge B(y))))),$$

onde R, Y e B são variáveis relacionais de ordem 2 e aridade 1, que representam as cores que serão usadas para colorir os vértices. Na sentença dizemos que todo elemento do domínio deve ter uma cor e que se dois vértices forem adjacentes, então eles terão cores diferentes.

Assim como temos a forma normal prenex para FO, que agrupa todos os quantificadores no início da fórmula, temos também algumas formas normais para HO^i. Dentre elas a *forma normal de Skolem generalizada*. Dizemos que uma fórmula φ de HO^i,

$i \geq 2$, está na forma normal de Skolem generalizada quando $\varphi = Q_1 V_1 \dots Q_k V_k \psi$, onde $Q_1, \dots, Q_k \in \{\exists, \forall\}$, $V_1, \dots, V_k$ são variáveis relacionais de ordem i e $\psi \in \mathrm{HO}^{i-1}$. As fórmulas de $\mathrm{HO}{=} \bigcup_i \mathrm{HO}^i$ que estão em forma normal de Skolem generalizada formam duas hierarquias bem conhecidas, cujos níveis são denotados por Σ^i_j e Π^i_j. A classe Σ^i_j compreende as fórmulas em HO^{i+1} que estão na forma normal de Skolem generalizada com no máximo $j-1$ alternâncias entre os blocos de quantificadores de variáveis de ordem $i+1$, começando por existencial. Cada bloco de quantificadores pode ter vários quantificadores do mesmo tipo, ou seja, um bloco de existenciais pode ter 1 ou mais quantificadores existenciais aninhados. A classe Π^i_j é definida dualmente.

Teorema 5.1.4. *[HT06] [HT03] Para todo $i \geq 2$ e para toda fórmula $\varphi \in \mathrm{HO}^i$, existem $\tilde{\varphi}$ e $\hat{\varphi}$ equivalentes a φ tal que $\tilde{\varphi} \in \Sigma^{i-1}_j$ e $\hat{\varphi} \in \Pi^{i-1}_j$.*

5.2 Expressividade

Na análise de expressividade de lógicas de ordem superior, vemos que bastante já foi estudado com relação à lógica de segunda-ordem. De fato, um dos resultados iniciais na área de complexidade descritiva relaciona um fragmento dessa lógica com a classe NP.

Teorema 5.2.1. *[Fag74] $\exists SO = NP$.*

Demonstração. **(a)** $\mathrm{NP} \subseteq \exists\mathrm{SO}$
Seja M uma máquina de Turing não-determinística que executa em tempo n^k para uma entrada $\mathcal{A}$, $|A| = n$. Escrevemos a seguinte sentença de segunda ordem

$$\Phi = \exists C_1^{2k} C_2^{2k} \dots C_g^{2k} \Delta^k \varphi$$

que diz "Existe uma computação de M que termina num estado de aceitação". Mais precisamente, a sentença de primeira ordem φ será tal que $(\mathcal{A}, \overline{C}, \Delta) \models \varphi$ sse $\overline{C}, \Delta$ é uma computação de M com estado final de aceitação. Vamos mostrar como codificar a computação de M.

$\overline{C}$ é uma matriz $\overline{C}(\overline{s}, \overline{t})$ com n^{2k} posições com espaço $\overline{s}$ e tempo $\overline{t}$ variando entre 0 e $n^k - 1$. Usaremos k-tuplas de variáveis $\overline{t} = t_1, \dots, t_k$ e $\overline{s} = s_1, \dots, s_k$ para codificar esses valores, onde cada t_i e s_i varia no domínio de $\mathcal{A}$. $\overline{C}(\overline{s}, \overline{t})$ codifica qual símbolo aparece na posição $\overline{s}$ da fita no tempo $\overline{t}$, para cada par $\overline{s}, \overline{t}$, se o cabeçote não estiver nessa posição. Se o cabeçote estiver na posição $\overline{s}$ então $\overline{C}(\overline{s}, \overline{t})$ codifica o par $\langle q, \sigma \rangle$, onde q é o estado de M no instante $\overline{t}$ e σ é o símbolo inscrito na fita. Seja $\Gamma = \{\gamma_0, \dots, \gamma_g\} = (Q \times \Sigma) \bigcup \Sigma$

a lista dos possíveis conteúdos que serão codificados por $\overline{C}$. Faremos C_i uma variável relacional $2k$-ária para $0 \leq i \leq g$. O significado intuitivo de $C_i(\overline{s}, \overline{t})$ é que a posição $\overline{s}$ no tempo $\overline{t}$ contém o símbolo γ_i. Podemos assumir que a cada passo, a máquina M fará um escolha dentre no máximo duas possíveis. Codificaremos essas escolhas na relação k-ária Δ. Intuitivamente, $\Delta(\overline{t})$ é verdade se no passo $\overline{t}+1$ foi feita a escolha "1", caso contrário foi feita a escolha "0".

Agora devemos escrever a sentença de primeira ordem $\varphi(\overline{C}, \Delta)$ que expressa que $\overline{C}, \Delta$ codifica uma computação de M com estado final de aceitação. A sentença consiste de quatro partes $\varphi = \alpha \wedge \beta \wedge \eta \wedge \zeta$, onde α afirma que o conteúdo inicial da fita é a codificação de $\mathcal{A}$, β afirma que nunca acontece de $C_i(\overline{s}, \overline{t})$ e $C_j(\overline{s}, \overline{t})$ serem ambos verdade para $i \neq j$, ou seja, dois símbolos não podem estar escritos na mesma posição no mesmo instante, η diz que para todo $\overline{t}$, a linha $\overline{t}+1$ é formada a partir da linha $\overline{t}$ e da escolha $\Delta(\overline{t})$ de M, e ζ diz que a última linha da matriz de computação inclui o estado de aceitação. De forma mais intuitiva, $\overline{C}, \Delta$ codificam a sequência de configurações que compõem a computação e a sentença φ verifica se essa sequência corresponde a uma possível computação da máquina M. Para mais detalhes de como descrever φ, ver [Imm99].

(b) $\exists$SO $\subseteq$ NP

Seja $\Phi = \exists R_1^{r_1} \ldots R_k^{r_k} \psi$ uma sentença de segunda ordem e σ o vocabulário de Φ. Queremos construir uma máquina de Turing M tal que para todo $\mathcal{A} \in \text{STRUC}[\sigma]$,

$$\mathcal{A} \models \Phi \Leftrightarrow M(\text{bin}(\mathcal{A})) \downarrow .$$

Seja $\mathcal{A}$ uma estrutura de entrada para M e $|A| = n$. A máquina M irá de forma não-determinística escrever uma *string* binária de tamanho n^{r_1} representando a R_1, e de forma similar para R_2 até R_k. Depois desse número polinomial de passos, teremos a seguinte estrutura expandida $\mathcal{A}' = (\mathcal{A}, R_1, R_2, \ldots, R_k)$. M aceitará sse $\mathcal{A}' \models \psi$. Esse teste pode ser feito com uma máquina na classe L [Imm99], então será facilmente feito por M. M aceita se existe uma escolha de relações $R_1, \ldots, R_k$ tal que $(\mathcal{A}, R_1, R_2, \ldots, R_k) \models \psi$. $\square$

Outro resultado importante envolvendo SO foi provado, estendendo o resultado de Fagin para toda a Hierarquia Polinomial.

Teorema 5.2.2. *[Sto77]* $\text{SO} = \bigcup_{j \geq 0} \Sigma_j^1 = \text{PH}$*, onde* PH *é a hierarquia polinomial.*

As lógicas com ordem superior a 2 não foram tão exploradas quanto SO, porém alguns resultados significativos já foram obtidos como, por exemplo, a prova de que a hierarquia de lógicas de ordem superior é estrita.

Teorema 5.2.3. *[HS91] Para todo* $i \geq 2$, $\mathrm{HO}^i \subset \mathrm{HO}^{i+1}$.

No resultado abaixo, obtemos um resultado análogo para os fragmentos existencial e universal.

Teorema 5.2.4. *[KV88] Para todo* $i \geq 2$,

1. $\exists\mathrm{HO}^i \subset \exists\mathrm{HO}^{i+1}$.
2. $\forall\mathrm{HO}^i \subset \forall\mathrm{HO}^{i+1}$.

Apesar da existência de tais resultados, apenas recentemente foi dada uma caracterização exata de cada fragmento da hierarquia de lógicas de ordem superior, exibida em [HT06] [HT03]. Esse resultado, que apresentaremos a seguir, se mostrou como uma extensão da correspondência estabelecida por Fagin-Stockmeyer entre os fragmentos de SO e os níveis da hierarquia polinomial. Para facilitar a apresentação dos resultados, definimos a função exp_i nos números naturais como sendo $exp_0(n) = n$ e $exp_i(n) = 2^{exp_{i-1}(n)}$, para todo $i \geq 1$.

Definição 5.2.5 (Hierarquia não-determinística em tempo exponencial)**.** Para todo $i \geq 0$ e $j \geq 1$, defina

1. $\mathrm{NEXPH}_i^0 = \bigcup_{c\in\mathbb{N}} \mathrm{NTIME}(exp_i(n^c))$
2. $\mathrm{NEXPH}_i^j = \mathrm{NEXPH}_i^{0^{\Sigma_{j-1}^p}}$, onde Σ_{j-1}^p é definido como na hierarquia polinomial.

Teorema 5.2.6. *[HT06] [HT03] Para todo* $i, j \geq 1$ *temos*

1. $\Sigma_j^i = \mathrm{NEXPH}_{i-1}^{j-1}$.
2. $\Pi_j^i = \mathrm{coNEXPH}_{i-1}^{j-1}$.

Pelo Teorema 5.3 temos que, para todo $i \geq 2$, $\mathrm{HO}^i = \bigcup_{j\geq 0} \Sigma_j^{i-1}$. Portanto, temos o seguinte corolário.

Corolário 5.2.7. *[HT06] Para todo* $i \geq 2$, $\mathrm{HO}^i = \bigcup_{j\geq 0} \mathrm{NEXPH}_{i-2}^j$

A classe de funções que podem ser computadas em tempo limitado por $exp_i(n^c)$ para algum $i \geq 0$ e para alguma constante natural c é conhecida como a classe de funções elementares de Kalmar.

Teorema 5.2.8. *[HS91] Seja $f(n)$ uma função elementar de Kalmar, então* $\bigcup_{i\geq 2} \mathrm{HO}^i = \mathrm{DTIME}(O(f(n)))$.

Pelo teorema 5.2.8 vemos que HO não é completa, ou seja, não pode expressar toda consulta computável. Por exemplo, a função $tow(n) = exp_n(1)$ é primitiva recursiva e, portanto computável. Mas máquina da classe DTIME$[exp_i(n^c)]$, para i fixo, pode computá-la.

Em [HT03] e [HT06] uma lógica de ordem superior, chamada Lógica de Ordem Variável (VO), foi definida. Essa lógica permite o uso de variáveis relacionais nas quais a ordem não é fixa, permitindo inclusive quantificar essa ordem. VO é capaz de expressar todos os problemas recursivos e, mais ainda, é também capaz de expressar consultas não-recursivas.

Teorema 5.2.9. *[HT06] Toda consulta recursiva pode ser expressa em* VO.

Teorema 5.2.10. *[HT06] Toda consulta não-recursiva pode ser expressa em* VO.

6 Lógicas de Ordem Superior com Operador de Menor Ponto Fixo

Como dito nos capítulos anteriores, é possível aumentar o poder expressivo de uma lógica usando alguns mecanismos. Neste capítulo analisaremos qual o ganho de expressividade quando, ao mesmo tempo, aumentamos a ordem da lógica e adicionamos o operador de menor ponto fixo. Além disso, investigaremos que classes de complexidade são caracterizadas por tais lógicas. Estes resultados estão em [FM10].

6.1 Sintaxe e Semântica

Como vimos no Capítulo 4, com uma fórmula de FO podemos definir uma função e, a partir dessa função, podemos construir uma nova relação. Do mesmo modo, usaremos lógicas de ordem superior para definir tais funções. Seja σ um vocabulário relacional, $\varphi(R, X_1, \ldots, X_k)$ uma fórmula de HO^i, para algum $i \geq 2$, R uma variável relacional de ordem $i+1$ e $X_1, \ldots, X_k$ variáveis relacionais de ordem i. De forma análoga ao que vimos para FO, para uma estrutura $\mathcal{A} \in \mathrm{STRUC}[\sigma]$, φ define uma função $F^{\varphi} : \mathcal{P}(A_i^k) \to \mathcal{P}(A_i^k)$, onde A_i^k é o produto cartesiano k vezes do conjunto A_i das relações de ordem i. Dessa forma, podemos definir o seguinte:

Definição 6.1.1 ($\mathrm{HO}^i(\mathrm{LFP})$)**.** Considere σ um vocabulário relacional, R uma variável relacional de aridade k e ordem $i+1$, $\varphi(R, X_1, \ldots, X_k)$ uma fórmula de $\mathrm{HO}^i(\mathrm{LFP})$, onde $X_1, \ldots, X_k$ são variáveis relacionais k-árias de ordem i, e $\mathcal{A} \in \mathrm{STRUC}[\sigma]$. A linguagem de $\mathrm{HO}^i(\mathrm{LFP})$ estende a linguagem de HO^i com a seguinte regra de formação:

- se $\varphi(R, X_1, \ldots, X_k)$ é positiva em R e $(V_1, \ldots, V_k)$ é uma tupla de relações de ordem i, então $[\mathbf{lfp}_{R, X_1, \ldots, X_k} \varphi(R, X_1, \ldots, X_k)](V_1, \ldots, V_k)$ é uma fórmula na qual as variáveis livres são $(V_1, \ldots, V_k)$.

A relação de satisfatibilidade de $\mathrm{HO}^i(\mathrm{LFP})$ estende a de HO^i com a seguinte definição:

- $\mathcal{A} \models [\mathbf{lfp}_{R,X_1,\ldots,X_k}\varphi(R, X_1, \ldots, X_k)](R_1, \ldots, R_k)$ sse $(R_1, \ldots, R_k) \in F^{\varphi}_{\infty}$.

Pela definição acima, vemos que ao utilizar o operador de menor ponto fixo poderemos definir algumas relações de ordem $i + 1$, mais especificamente relações indutivas, dentro de HO^i. Parece claro, portanto, que há um ganho de expressividade ao adicionarmos esse operador, o que confirmaremos na próxima seção.

6.2 Expressividade

Nesta seção investigaremos o poder expressivo de HO^i(LFP), ou seja, a adição do operador **lfp** à HO^i. Como sabemos existem outros operadores de ponto fixo e em [ST06] os autores exploram a adição dos operadores **ifp** e **pfp** à HO^i. Eles mostram os seguintes teoremas:

Teorema 6.2.1. *O ponto fixo inflacionário em HO^i, para $i \geq 2$, pode ser expresso em $\exists HO^{i+1}$, ou seja, $HO^i(IFP) \subseteq HO^{i+1}$.*

Teorema 6.2.2. *O ponto fixo parcial em HO^i, para $i \geq 2$, pode ser expresso em $\exists HO^{i+2}$, ou seja, $HO^i(PFP) \subseteq HO^{i+2}$.*

Ao adicionar o operador de menor ponto fixo, obtemos resultados semelhantes a esses como corolários diretos de nosso principal resultado que mostra qual classe de complexidade é capturada por HO^i(LFP). Tal caracterização não é dada no artigo acima citado.

O teorema a seguir mostra qual classe de complexidade é capturada pelo primeiro nível de HO^i(LFP). Apesar deste teorema ser citado em [Imm99], não encontramos nenhuma prova ou referência para tal prova.

Teorema 6.2.3. *Sobre estruturas fnitas e ordenadas,* SO(LFP) *captura* EXP*, a classe dos problemas decididos em tempo determinístico exponencial, ou seja,* SO(LFP) = EXP*.*

Com a intenção de provar o teorema acima, nós obtivemos um resultado mais geral. Ao invés de caracterizar apenas o primeiro nível, vimos que era possível caracterizar todos os níveis da hierarquia. De fato, provamos que cada nível dessa hierarquia captura um nível da hierarquia determinística em tempo exponencial, como veremos a seguir.

Definição 6.2.4. Seja i-EXP = TIME($exp_i(n^k)$), para todo k. A hierarquia determinística de tempo exponencial é definida como: EXPH = $\bigcup_{i \geq 1} i$-EXP.

Teorema 6.2.5. $\mathrm{HO}^i(\mathrm{LFP}) = (i-1)\text{-EXP}$, *para todo* $i \geq 2$.

Demonstração. **(a)** $\mathrm{HO}^i(\mathrm{LFP}) \subseteq (i-1)\text{-EXP}$.
Por indução nas fórmulas φ. Toda fórmula atômica de $\mathrm{HO}^i(\mathrm{LFP})$ é uma fórmula sem o operador **lfp**, logo é uma fórmula de HO^i. Pelo Corolário 5.10, temos que φ pode ser avaliada por uma máquina em $\bigcup_{j\geq 0} \mathrm{NEXPH}^j_{i-2}$ e assim por uma máquina em $(i-1)$-EXP.

$\varphi = \neg\psi$. Pela hipótese indutiva, existe uma máquina $M_\psi \in (i-1)$-EXP que avalia ψ. A máquina M_φ usa a máquina M_ψ para verificar se $\mathcal{A} \models \psi$. Se M_ψ aceita, então M_φ rejeita e vice-versa.

$\varphi = \psi_1 \wedge \psi_2$. Sejam M_{ψ_1} e M_{ψ_2} as máquinas que avaliam as fórmulas ψ_1 e ψ_2. A máquina M_φ passará como entrada para essas máquinas a codificação da estrutra. Se ambas aceitarem, então M_φ aceitará. Caso contrário, ela rejeitará.

$\varphi = \psi_1 \vee \psi_2$. Análogo ao caso anterior, *mutatis mutandis*.

$\varphi = \exists x\psi$. Pela hipótese indutiva, existe uma máquina $M_\psi \in (i-1)$-EXP que avalia ψ. Perceba que ψ tem apenas uma variável livre, no caso x. Assim, a máquina M_φ escreve na sua fita de trabalho a codificação binária de $j = 0, \ldots, n-1$ e, usando M_ψ, verifica se $\mathcal{A} \models \psi(j/x)$. Se M_ψ aceita, para algum j, então M_φ aceita. Caso contrário, M_φ rejeita. Como o tamanho do domínio é n, tal processo será executado no máximo n vezes e, portanto, $M_\varphi \in (i-1)$-EXP.

$\varphi = \forall x\psi$. Análogo ao caso anterior, *mutatis mutandis*.

$\varphi = \exists X\psi$. X é uma variável relacional de ordem j, $2 \leq j \leq i$, e aridade k. Pela hipótese indutiva, existe uma máquina $M_\psi \in (i-1)$-EXP. Perceba que ψ tem uma variável livre X. M_φ usa uma fita de trabalho para codificar as relações de ordem j e aridade k. Essa fita usa $exp(i-2, n^k)$ células e pode codificar $exp(i-1, n^k)$ relações diferentes. Para cada relação R descrita na fita, M_ψ é usada para verificar se $\mathcal{A} \models \psi(R)$. Se M_ψ aceita para algum R, então M_φ aceita. Caso contrário, M_φ rejeita. Como são feitas no máximo $exp(i-1, n^k)$ chamadas a M_ψ, então M_φ executará em tempo $exp(i-1, n^k).exp(i-1, O(n^c)) = exp(i-1, O(n^c))$. Assim, $M_\varphi \in (i-1)$-EXP.

$\varphi = \forall X\psi$. Análogo ao caso anterior, *mutatis mutandis*.

$\varphi = [\mathbf{lfp}_{R,X_1,\ldots,X_k}\psi(R, X_1, \ldots, X_k)](R_1, \ldots, R_k)$. Pela hipótese indutiva, existe $M_\psi \in (i-1)$-EXP. A máquina M_φ usa duas fitas de trabalho para escrever a codificação das relações de ordem $i+1$ que serão atribuídas a variável relacional R. Como essas relações são de ordem $i+1$, precisamos de $exp(i-1, n^k)$ células para codificá-las. No primeiro

passo, $R = \emptyset$ é a relação codificada na fita 1. Usando a máquina M_ψ, M_φ computa a seguinte relação: $R' = \{(V_1, \ldots, V_k) \mid \mathcal{A} \models \psi(R, V_1, \ldots, V_k)\}$. Se $R = R'$ então o ponto fixo foi alcançado e é suficiente testar se $(X_1, \ldots, X_k) \in R$. Caso contrário, a máquina copia o conteúdo da fita 1 na fita 2, apaga o conteúdo da fita 1 e computa R' novamente. Como ψ é positiva em R, então sabemos que o ponto fixo é alcançado no máximo em $exp(i-1, n^k)$ passos. Assim, a máquina M_ψ será chamada no máximo $exp(i-1, n^k)$ vezes e M_φ executará em tempo $exp(i-1, n^k).exp(i-1, O(n^c)) = exp(i-1, O(n^c))$. Portanto, $M_\varphi \in (i-1)$-EXP.

(b) $(i-1)$-EXP $\subseteq$ HO^i(LFP).

Seja $M \in (i-1)$-EXP uma máquina de Turing que executa em tempo $exp(i-1, n^k)$, para algum k. Podemos concluir que M usa no máximo $exp(i-1, n^k)$ células de sua fita de trabalho. Queremos definir uma fórmula φ tal que M aceita $\mathcal{A}$ sse $\mathcal{A} \models \varphi$. Nossa fórmula descreve a computação da máquina e, como a computação da máquina pode ser descrita como uma sequência de configurações, usaremos uma relação C de aridade 3 para descrever tal sequência. A configuração de uma máquina é definida pelo seu estado corrente, a posição do cabeçote e o conteúdo das fitas de trabalho. Usaremos a primeira componente como um temporizador, que indicará em qual passo da computação estaremos. Como a máquina executa em tempo $exp(i-1, n^k)$, usaremos uma relação de ordem i para codificar o temporizador. O segundo indicará qual o tipo de informação está codificado naquela tupla, por exemplo, se indica o estado corrente. Como só temos 3 possíveis informações, poderíamos utilizar variáveis individuais, facilmente com uma variável relacional de ordem i. A terceira componente indica a informação propriamente dita, por exemplo, qual o conteúdo da fita de trabalho j. Nesse caso, quando indicamos a posição do cabeçote ou o conteúdo de uma fita de trabalho, precisamos codificar uma quantidade exponencial de posições, como mencionado anteriormente. Portanto, usaremos relações de ordem i novamente. Como as componentes da relação C são de ordem i, então C é uma relação de ordem $i+1$. Agora, utilizamos o operador de menor ponto fixo para definir C, usando as instruções da máquina para estabelecer a fórmula ψ tal que $C = [\mathbf{lfp}_{R,X_1,X_2,X_3}\psi(R, X_1, X_2, X_3)]$. Assim, é suficiente verificar se existe uma tupla cuja primeira componente indique que o tempo é $exp(i-1, n^k)$, a segunda componente indique que a informação é o estado corrente e a terceira componente indica que o estado é de aceitação. □

O Teorema 6.2.3 é um corolário direto do Teorema 6.2.5. O último segue o mesmo argumento de resultados conhecidos que foram citados em todo texto, diferindo deles

nos detalhes técnicos da demonstração. Um fato interessante que podemos observar é a estreita relação entre a adição do operador de menor ponto fixo e o ganho exponencial de tempo. Por exemplo, FO captura a hierarquia logarítmica e, quando adicionamos o operador de menor ponto fixo para obter FO(LFP), a nova lógica captura a classe P, i.e., um ganho exponencial em tempo com relação a LH. Esse ganho também fica evidente no Teorema 6.2.5. A seguir, alguns corolários que seguem quase que diretamente do nosso resultado principal.

Corolário 6.2.6. $\mathrm{HO}^i(\mathrm{LFP}) \subset \mathrm{HO}^{i+1}(\mathrm{LFP})$*, para todo* $i \geq 2$*.*

Corolário 6.2.7. $\mathrm{HO}^i(\mathrm{LFP}) \subseteq \mathrm{HO}^{i+1}$*, para todo* $i \geq 2$*.*

Corolário 6.2.8. $\bigcup_{i\geq 2} \mathrm{HO}^i(\mathrm{LFP}) = \bigcup_{i\geq 2} \mathrm{HO}^i$*.*

O Corolário 6.2.6 prova que a hierarquia $\mathrm{HO}^i(\mathrm{LFP})$ não colapsa. De fato, a prova desse corolário é consequência do Teorema 6.2.5 e do Teorema da Hierarquia de Tempo, Teorema 2.8. O Corolário 6.2.7 é similar ao teorema provado em [ST06] para $\mathrm{HO}^i(\mathrm{IFP})$, e o Corolário 6.2.8 estabelece que não há ganho de expressividade quando consideramos a união de todos os níveis da hierarquia.

7 Conclusão

A área de Complexidade Descritiva visa caracterizar a complexidade de um problema por um viés independente da máquina, através da expressividade de uma linguagem capaz de definir uma classe de problemas. Muitos resultados foram obtidos envolvendo FO e suas extensões, bem como classes de complexidade polinomiais. De modo geral, nesse trabalho investigamos a expressividade da lógica modal básica e das lógicas temporais CTL e CTL*. Investigamos também as lógicas de ordem superior e qual o ganho de expressividade ao adicionar o operador de menor ponto fixo a elas.

No capítulo inicial, revisamos a área de complexidade computacional, onde definimos conceitos importantes como máquina de Turing, problema de decisão e classes de complexidade, além de apresentar alguns teoremas importantes da área. Na revisão de complexidade descritiva, definimos as consultas booleanas e explicamos a noção de uma lógica capturar uma classe de complexidade. Apresentamos também alguns resultados como FO=LH e que as classes não-determinísticas em espaço são fechadas para o complemento.

Em seguida investigamos a linguagem modal básica e seu poder expressivo. Vimos que as lógicas K, S4, e S5 não são suficientes para expressar o problema REACH, pois nenhuma delas é mais expressiva que FO. Já no caso das lógicas CTL e CTL*, vemos que os operadores temporais e os quantificadores de caminho aumentam seu poder expressivo de forma que REACH pode ser expresso. Estes resultados estão publicados em [FM09].

Antes de abordar o tema principal, revisamos as lógicas com operadores de ponto fixo e as lógicas de ordem superior. No capítulo em que falamos das lógicas com operador de ponto fixo, estudamos a adição de três tipos de ponto fixo: o menor, o inflacionário e o parcial e as lógicas que deles se originam que são FO(LFP), FO(IFP) e FO(PFP), respectivamente. Vimos que as duas primeiras possuem o mesmo poder expressivo e que o ganho com relação a FO é significativo e, no caso de FO(PFP), podemos dizer que tem pelo menos mesmo poder expressivo, mas não sabemos se FO(LFP) $<$ FO(PFP).

No capítulo de lógica de ordem superior, inicialmente apresentamos a sintaxe e semântica das lógicas. Exibimos alguns dos resultados mais conhecidos com relação a expressividade e vemos que essas lógicas caracterizam a hierarquia não-determinística exponencial e não a classe de todos os problemas recursivos, como se pensava. Apesar disso, exibimos uma lógica que tem poder expressivo maior que HO^i e consegue expressar, além dos problemas recursivos, os problemas recursivamente enumeráveis.

Como contribuição investigamos qual o ganho de expressividade quando unimos, numa mesma lógica, relações de ordem superior e o operador de menor ponto fixo. A idéia de adicionar algum ponto fixo a lógicas de ordem superior foi formulada inicialmente em [ST06]. Eles mostraram resultados de expressividade semelhantes aos nossos para os pontos fixos inflacionário e parcial, mas não caracterizam nenhuma classe de complexidade com essas lógicas. Nosso resultado principal estabelece que $\mathrm{HO}^i(\mathrm{LFP})$ captura a classe de complexidade $(i-1)$-EXP, para todo $i \geq 2$. Dessa forma, nós demos uma caracterização para todos os níveis da Hierarquia Determinística em Tempo Exponencial. A partir desse resultado, obtemos diretamente que SO(LFP)=EXP, que é um teorema bastante citado na literatura para o qual não encontramos prova, e alguns corolários interessantes. O primeiro estabelece que é suficiente aumentar a ordem das variáveis em 1 para capturar o operador de menor ponto fixo, ou seja, $\mathrm{HO}^i(\mathrm{LFP}) \subseteq \mathrm{HO}^{i+1}$, para todo $i \geq 2$. Também provamos que a hirerarquia $\mathrm{HO}^i(\mathrm{LFP})$ não colapsa, pois cada nível é mais expressivo do que o anterior e, finalmente, que apesar do ganho de expressividade em cada nível da hierarquia $\mathrm{HO}^i(\mathrm{LFP})$, não há ganho quando consideramos a união de todos os níveis, o que pode ser expresso por $\bigcup_{i\geq 2} \mathrm{HO}^i(\mathrm{LFP}) = \bigcup_{i\geq 2} \mathrm{HO}^i$.

APÊNDICE A – Lógicas K, S4 e S5

Nesse apêndice apresentaremos as lógicas modais K, S4 e S5 e veremos que podemos expressar os teoremas de tais lógicas com fragmentos de FO. Para fazer tal análise, usaremos a noção de tradução já apresentada, mas com as modificações necessárias.

Definição A.0.9 (Lógica K)**.** *A lógica* K é o menor conjunto de fórmulas que contém todas as tautologias, o axioma K $\Box(p \to q) \to (\Box p \to \Box q)$, e é fechado para *modus ponens*, susbtituição uniforme e generalização.

Teorema A.0.10. *Toda fórmula de* K *pode ser traduzida para uma fórmula em* FO^2*.*

Demonstração. Pela Definição 3.1.7 de tradução padrão e como a linguagem de K é a modal básica, o resultado segue direto pelo Teorema 3.1.9. □

Definição A.0.11 (Lógica S4)**.** A *lógica* S4 é o menor conjunto de fórmulas que contém todas as tautologias, o axioma K, o axioma T $p \to \Diamond p$, o axioma 4 $\Diamond\Diamond p \to \Diamond p$, e é fechado para *modus ponens*, substituição uniforme e generalização.

Definição A.0.12 (Lógica S5)**.** A *lógica* S5 é o menor conjunto de fórmulas que contém todas as tautologias, o axioma K, o axioma T, o axioma 4, o axioma B $p \to \Box\Diamond p$, e é fechado para *modus ponens*, substituição uniforme e generalização.

Os teoremas de S4 são válidos na classe dos modelos que tem como relação de acessibilidade uma relação reflexiva e transitiva imposta pelos axiomas T e 4, respectivamente. No caso de S5, os teoremas são válidos na classe de modelos reflexivos, transitivos e simétricos e a simetria é consequência do axioma B. Chamaremos as classe de modelos citadas acima de $\mathfrak{M}_{S4}$ e $\mathfrak{M}_{S5}$, respectivamente.

Utilizando a tradução padrão para verificar se S4 e S5 podem ser expressos em FO^2, observamos que não é possível preservar a validade nos modelos , ou seja, as fórmulas modal e de primeira ordem não seriam equivalente. Com isso, poderíamos ter algum

modelo que está $\mathfrak{M}_{S4}$ ou $\mathfrak{M}_{S5}$ satisfazendo um teorema de S4 ou S5. No caso da lógica K, o Teorema 3.1.8 mostra que a validade é preservada. Dessa forma, precisamos definir uma tradução para S4 e S5 que preserve a validade na classe de modelos $\mathfrak{M}_{S4}$ e $\mathfrak{M}_{S5}$.

Definição A.0.13 (Tradução para S4)**.** A *tradução para* S4, ST_x^4, é semelhante à tradução padrão da Definição 3.1.7, alterando apenas o seguinte caso:

1. $ST_x^4(\Diamond\varphi) = \exists y(Rxy \wedge ST_y^4(\varphi)) \wedge (\forall x Rxx) \wedge (\forall x \forall y \forall z(Rxy \wedge Ryz \rightarrow Rxz))$, onde y é uma variável que ainda não foi usada na tradução.

Teorema A.0.14. *Seja φ uma fórmula de* S4*. Temos que:*

1. *Para todo modelo M e todo w em M: $M \in \mathfrak{M}_{S4}$ e $M, w \models \varphi$ sse $M \models ST_x^4(\varphi)[w]$.*
2. *Para todo modelo M: $M \in \mathfrak{M}_{S4}$ e $M \models \varphi$ sse $M \models \forall x ST_x^4(\varphi)$.*

Demonstração. A prova deste teorema é semelhante à prova para o Teorema 3.1.8 apresentada em [BdRV01]. Precisamos só garantir que os modelos que satisfazem as fórmulas traduzidas sejam reflexivos e transitivos, e isto é garantido adicionando-se à tradução padrão as fórmulas que impõem a reflexividade e transitividade de R. □

Teorema A.0.15. *Toda fórmula de* S4 *pode ser traduzida para uma fórmula de* FO^3*.*

Demonstração. Na tradução da Definição A.0.13, expressamos a propriedade de um modelo ser reflexivo e transitivo. Para expressar reflexividade basta uma variável, mas para expressar a transitividade são necessárias três variáveis. Como apenas isso diferencia essa tradução da tradução padrão da Definição 3.1.7 e, pelo Teorema 3.1.9, temos que só precisamos de três variáveis para expressar S4. □

Para S5 a situação é diferente, pois podemos assumir duas classes de modelos para S5: os modelos cuja relação de acessibilidade é de equivalência e os modelos cuja relação é universal [Che80].

Definição A.0.16 (Tradução para S5 com relação de equivalência)**.** A *tradução para* S5 *com relação de equivalência*, $ST_x^{5.1}$, é semelhante à tradução padrão da Definição 3.1.7, alterando apenas o seguinte caso:

1. $ST_x^{5.1}(\Diamond\varphi) = \exists y(Rxy \wedge ST_y^{5.1}(\varphi)) \wedge (\forall x Rxx) \wedge (\forall x \forall y \forall z(Rxy \wedge Ryz \rightarrow Rxz)) \wedge (\forall x \forall y(Rxy \rightarrow Ryx))$, onde y é uma variável que ainda não foi usada na tradução.

Teorema A.0.17. *Seja φ uma fórmula de* S5*. Temos que:*

1. *Para todo modelo M e todo w em M: $M \in \mathfrak{M}_{S5}$ e $M, w \models \varphi$ sse $M \models ST_x^{5.1}(\varphi)[w]$.*
2. *Para todo modelo M: $M \in \mathfrak{M}_{S5}$ e $M \models \varphi$ sse $M \models \forall x ST_x^{5.1}(\varphi)$.*

Demonstração. Semelhante à prova do Teorema A.0.14. □

Teorema A.0.18. *Toda fórmula de* S5 *pode ser traduzida para uma fórmula de* FO^3*.*

Demonstração. Semelhante à prova do Teorema A.0.15. □

Definição A.0.19 (Tradução para S5 com relação universal)**.** A *tradução para S5 com relação universal*, $ST_x^{5.2}$, é semelhante à tradução padrão da Definição 3.1.7, alterando apenas o seguinte caso:

1. $ST_x^{5.2}(\Diamond\varphi) = \exists x ST_x^{5.2}(\varphi)$.

Teorema A.0.20. *Seja φ uma fórmula de* S5 *e $\mathfrak{M}_U$ a classe de modelos com relação de acessibilidade universal. Temos que:*

1. *Para todo modelo M e todo w em M: $M \in \mathfrak{M}_U$ e $M, w \models \varphi$ sse $M \models ST_x^{5.2}(\varphi)[w]$*
2. *Para todo modelo M: $M \in \mathfrak{M}_U$ e $M \models \varphi$ sse $M \models \forall x ST_x^{5.2}(\varphi)$*

Demonstração. A prova deste teorema é semelhante à prova para o Teorema 3.1.8 apresentada em [BdRV01]. Basta ver que, ao varrer o domínio W com o quantificador existencial, qualquer elemento que garantir a satisfação de φ vai estar relacionado com x, pois a relação de acessibilidade é universal. Isto é garantido pela tradução da Definição A.0.19. □

Teorema A.0.21. *Toda fórmula de* S5 *pode ser traduzida para uma fórmula de* FO^1*.*

Demonstração. Como a relação de acessibilidade é universal, não é necessário considerá-la explicitamente na tradução. Dessa forma, só precisamos de uma variável. □

Referências

[BdRV01] Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal Logic*. Cambridge University Press, 2001.

[CGP01] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, 3rd edition, 2001.

[Che80] Brian F. Chellas. *Modal logic: an introduction*. Cambridge University Press, 1980.

[DSW94] Martin D. Davis, Ron Sigal, and Elaine J. Weyuker. *Computability, Complexity, and Languages: fundamentals of Theoretical Computer Science*. Academic Press, 2nd edition, 1994.

[EF95] Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite Model Theory*. Springer, 1995.

[EFT94] Heinz-Dieter Ebbinghaus, Jörg Flum, and Wolfgang Thomas. *Mathematical Logic*. Springer, 2nd edition, 1994.

[Fag74] Ronald Fagin. Generalized first-order spectra and polynomial-time recognizable sets. In *Complexity of computation*, volume 7, pages 43–73. AMS Bookstore, 1974.

[FM09] Cibele Matos Freire and Ana Teresa Martins. Lógicas modais e complexidade descritiva. *Anais do Encontro Nacional de Inteligência Artificial*, VII:1099–1108, Julho 2009.

[FM10] Cibele Matos Freire and Ana Teresa Martins. The descriptive complexity of the deterministic time hierarchy. Aceito no LSFA - Workshop on Logical and Semantic Frameworks, with Applications, 2010.

[HS65] J. Hartmanis and R. E. Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117:285–306, 1965.

[HS91] Richard Hull and Jianwen Su. On the expressive power of database queries with intermediate types. *Journal of Computer and System Sciences*, 43:219–267, 1991.

[HT03] Lauri Hella and José María Turull Torres. Expressibility of higher order logics. *Eletronic Notes in Theoretical Computer Science*, 84:129–140, 2003.

[HT06] Lauri Hella and José María Turull Torres. Computing queries with higher order logics. *Theoretical Computer Science*, pages 197–214, 2006.

[Imm88] Neil Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on Computing*, 17(5):935–938, 1988.

[Imm99] Neil Immerman. *Descriptive Complexity*. Springer, 1999.

[KV88] Gabriel M. Kuper and Moshe Y. Vardi. On the complexity of queries in the logical data model. *Lecture Notes in Computer Science*, pages 267–280, 1988.

[Lib04] Leonid Libkin. *Elements of Finite Model Theory*. Springer, 2004.

[Pap94] Christos H. Papadimitriou. *Computational Complexity*. Addison Wesley Logman, 1994.

[ST06] Klaus-Dieter Schewe and José María Turull Torres. Fixed-point quantifiers in higher order logics. In *Proceeding of the 2006 conference on Information Modelling and Knowledge Bases XVII*, pages 237–244. IOS Press, 2006.

[Sto77] Larry J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3:1–22, 1977.

[Sze88] Róbert Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26(3):279–284, 1988.